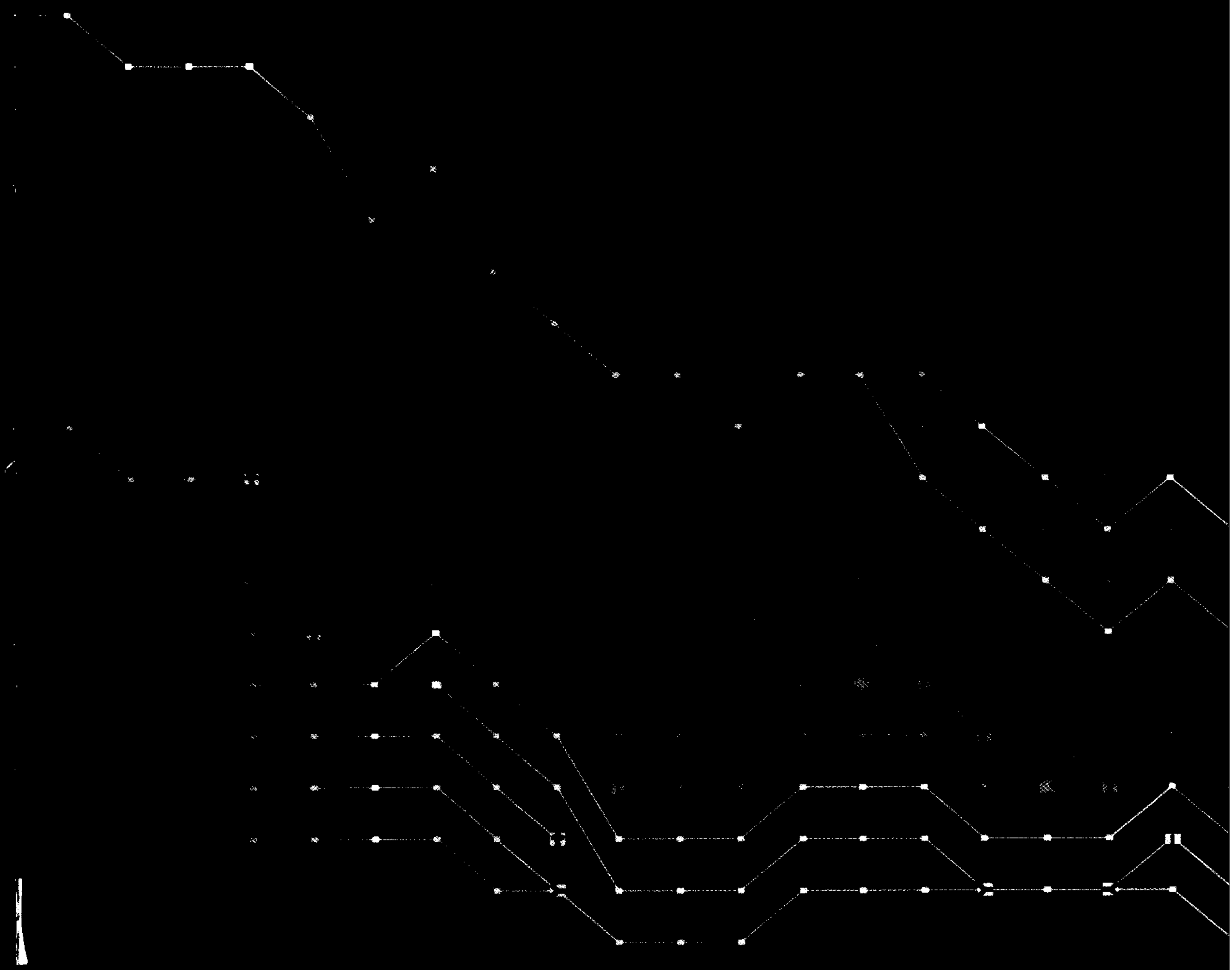
Feature-Based Visualization of Time-Dependent Data

Freek Reinders



TR 3662 S

Stellingen

behorende bij het proefschrift

Feature-Based Visualization of Time-Dependent Data

Freek Reinders

12 maart 2001

1. Feature extractie is de enige praktische manier om zeer grote data sets visueel te analyseren.

2. Feature extractie richt de aandacht van de onderzoeker minder op de data en meer op de onderzochte verschijnselen.

3. Het nauwkeurig definiëren van features en events en de bepaling van de detectie criteria, behoren tot het specialisme van de gebruiker en moet als een deel van diens onderzoek worden beschouwd.

4. Ondanks dat allerlei methoden toepasbaar zijn in beide vakgebieden, zijn beeldverwerking en data visualisatie zeer verschillend.

5. De toename aan verwerkingskracht van computers leidt wellicht tot het oplossen van het tera-byte probleem, maar leidt zeker ook tot een nieuw peta-byte probleem.

6. Wandelen in de bergen is een goede metafoor voor het leven; de volgende top lijkt altijd hoger, het volgende dal dieper, maar meestal is het de rust die je zoekt.

7. Het openzetten van bruggen in Den Bosch naar aanleiding van supporters rellen, geeft aan dat middeleeuwse methoden ter bestrijding van geweld nog steeds effectief zijn.

8. Real-life soap programma's zoals Big Brother en De Bus zijn het levende bewijs dat het nieuws daar is waar de camera's staan.

9. De nieuwswaarde van een ramp hangt af van het aantal doden, de afstand tot je huis en de technologische ontwikkeling van het getroffen land.

10. Voor roeiers zijn de beste stellingen de rolstellingen.

1. Feature extraction is the only practical way to visually analyze large data sets.

2. Feature extraction directs the attention of the researcher less to the data and more to the phenomena under investigation.

3. The exact definition of features and events, and the determination of the detection criteria, belong to the expertise of the user and should be considered part of his research.

4. Despite the fact that many techniques are applicable in both fields, image processing and data visualization are very different.

5. The increase of the computational power of computers may lead to the solution of the tera-byte problem, but it will certainly lead to a new peta-byte problem.

6. Hiking in the mountains is a good metaphor for life; the next peak always seems higher, the next valley deeper, but most of the time it is peace that you seek.

7. The raising of the bridges in Den Bosch in response to the supporter's riots, shows that medieval methods are still effective for the prevention of violence.

8. Real-life soap programs such as Big Brother and De Bus are the living proof that news is there where the cameras are.

9. The news value of a disaster depends on the number of casualties, the distance to your home, and the technological development of the country hit by the disaster.

10. For rowers the best 'stellingen' are the 'rolstellingen'.

[illegible]

TR 3662

Feature-Based Visualization of Time-Dependent Data

About the cover background
A visualization of an event graph, which gives an overview of feature evolutions in time-dependent data.

About the back cover
Top left image: skeleton shape reconstruction of vortex structures in a turbulent flow, see Figure 8.17. Data courtesy D. Silver and X. Wang of Rutgers University.

Top right image: iconic representation of the skeleton graph for one frame in the time-series of the flow with turbulent vortex structures, see Figure 5.15. Data courtesy D. Silver and X. Wang of Rutgers University.

Bottom left image: one step in the process of feature tracking, see Figure 6.4. Data generated by our synthetic data generator.

Bottom right image: 3D vortices in the flow past a tapered cylinder, see Figure 8.28. Data courtesy NASA Ames Research Center.

SION NWO

This work is supported by the Netherlands Computer Science Research Foundation (SION), with financial support of the Netherlands Organization for Scientific Research (NWO).

Advanced School for Computing and Imaging

This work was carried out in graduate school ASCI.
ASCI dissertation series number 61.

Feature-Based Visualization of Time-Dependent Data

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.F. Wakker,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 12 maart 2001 om 16:00 uur
door

Koob Frederik Jan REINDERS

natuurkundig ingenieur
geboren te Noordoostpolder

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

Toegevoegd promotor:
Ir. F.H. Post

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof.dr.ir. F.W. Jansen,	Technische Universiteit Delft, promotor
Ir. F.H. Post,	Technische Universiteit Delft, toegevoegd promotor
Prof.dr.ir. R.L. Lagendijk,	Technische Universiteit Delft
Prof.dr.ir. F.T.M. Nieuwstadt,	Technische Universiteit Delft
Prof.dr.ir. F.C.A. Groen,	Universiteit van Amsterdam
Prof.dr. D. Silver,	Rutgers University
Dr. H.J.W. Spoelder,	Vrije Universiteit Amsterdam

Preface

The research described in this thesis was carried out at the Computer Graphics & CAD/CAM group of Delft University of Technology. It is the fifth PhD project in a series of projects on visualization. The project was supervised by Frits Post (TU Delft) and Hans Spoelder (VU Amsterdam), and was supported by the Netherlands Computer Science Research Foundation (SION), with financial support of the Netherlands Organization for Scientific Research (NWO).

The research concentrated on feature-based visualization techniques. Feature-based visualization is particularly suited to visualize large data sets, such as time-dependent data. It directs the attention to interesting phenomena (features) in the data; the objective is to extract, quantify, and visualize these features. Feature-based visualization of time-dependent data focuses on the evolution of features in time. The process consists of four steps: feature extraction, feature tracking, event detection, and visualization. These four steps are investigated in depth in this thesis.

This thesis is accompanied by a CD-ROM that contains animations, images, sources, and web pages. It is included because time-dependent data cannot be visualized only on paper: animations are indispensable. It is strongly recommended to reproduce the CD, and give it to everybody who is interested.

A large number of people is of importance for the completion of a dissertation. They contribute in many different ways; scientifically, technically, and personally. It is impossible to thank everybody in person, but there are some people I would like to thank in particular.

First of all I would like to thank my direct supervisor Frits Post, you were a great inspiration during my research. Together with Hans Spoelder, my second supervisor, you were able to put me on the right track a couple of times. The two of you form a great pair of supervisors; Frits with his ideas and visions that seem to be never-ending, and Hans with his background in physics and his practice in implementation.

Two graduate students contributed to the work in a huge fashion: Melvin Jacobson and Benjamin Vrolijk. To Melvin I would like to say: I'm glad you graduated before me, because now I have a wonderful chapter about skeletons in my thesis. Benjamin: I wish you had started your graduation earlier, because

you added many useful things. Unfortunately, there was no time to describe them all in my thesis, I hope you can write them down in your own.

Another person that I would like to thank personally is Ari Sadarjoen, who has been my office mate for three years. You contributed in both scientific and personal ways. It is funny to see how two people that have such different lives, can get along very well. I sincerely hope you will recover completely from your illness.

A number of people were involved with the project indirectly, but they also deserve credits. My promotor Erik Jansen for giving the opportunity to do this project and for quickly, yet accurately, reading the thesis. Jack van Wijk with his simple questions that are hard to answer, but always seem to give you new insight. Deborah Silver and her group who gave me the data of the useful application with the turbulent vortex structures. Pieter Jonker for making the skeleton algorithms available to us.

Then, I would like to thank all people at the CG & CC group: the (PhD) students, the technical staff, and the teachers. They created a pleasant environment to work in, and an atmosphere which I enjoyed very much.

Also, I would like to thank all friends from the rowing club DDS, who made sure that besides work there is also time for recreation, although time flies when you're having fun.

Finally, I am grateful to my parents for their love and unconditional support; they gave me the feeling that there is always someone to fall back to.

Thanks to you all,

Freek Reinders

Contents

Chapter 1

Introduction

1.1 Visualization

Visualization is the process that transforms (numerical) data into a visual representation. Numerical data are a low-level representation of natural phenomena, and are often hard to interpret directly. Visual representations are much easier to understand for humans. The human eye has phenomenal capabilities for detecting structures, shapes and patterns. The expression is that 'an image says more than a thousand words', or numbers in our case.

Examples from our daily life are the images of the weather forecast, as in Figure 1.1. These images are widely known; we see them on TV and we know more or less how to interpret them. Figure 1.1 tells us that there is an area of low pressure to the northwest of Great-Britain. From experience, we know that this means a high probability of rain in the area of Western-Europe.

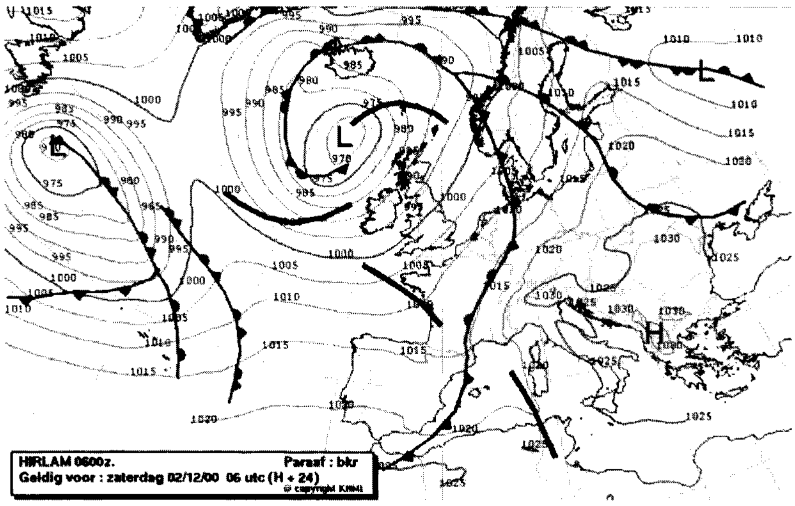


Figure 1.1: A visualization of a weather forecast, with curves showing iso-pressure lines, and icons showing fronts, and high and low pressure areas. Image courtesy the KNMI in The Netherlands.

Figure 1.1 is a good example of a feature-based visualization. Although the original data includes wind, pressure, and temperature values over the whole continent, the focus is directed to the interesting features: areas of locally high and low pressure values, and the fronts. This is exactly the objective of feature-based visualization: to extract and visualize the significant phenomena in the data. Feature-based visualization often results in schematic, symbolic representations such as glyphs or icons, for instance the H and L signs, and the front lines with motion indicators in the figure.

Once the features have been identified, it is interesting to see how they behave in time. If the low pressure area moves towards Scandinavia, we can expect rain for the next couple of days. Meteorologists build computer models to predict tomorrow's weather. These models result in time-dependent data in which each time step can be represented by a visualization such as in Figure 1.1. The images show us the evolution of the features, and we may recognize interesting stages in this evolution by visual inspection, such as: will this storm depression pass over our region, and when will this happen?

The example of the weather illustrates the need for *feature-based visualization of time-dependent data*. In general, features can be any interesting pattern, structure, or object that is considered relevant for the investigation of an application. In previous work [van Walsum, 1995] a generic feature extraction technique was developed based on selective and iconic visualization. Here, we want to further analyze the features, for instance to investigate their behavior in time.

1.2 Objectives

The main objective of the research described in this thesis is to find ways to analyze the behavior of features in time. This can be achieved in four steps: feature extraction, feature tracking, event detection, and visualization.

1. **Feature extraction.** In each time-step (frame) extract the features, calculate and store their attributes. A correspondence problem remains between features in successive frames.

2. **Feature tracking.** Solve the correspondence problem based on the attribute descriptions of the features. Feature tracking finds continuous paths of features through time.

3. **Event detection.** Detect certain interesting 'events' by investigating the evolution of features. Examples of events are birth, death, split, and merge.

4. **Visualization.** Interactively visualize the evolution of features in a player, show the relations between features in successive frames, and highlight particular events.

The basis for this process are the numerical attributes which describe the characteristics of features. Therefore, part of the research focuses on the *quantification* of features.

One objective is to establish an accurate and stable basis of feature attributes for temporal coherence. It is important to know if the calculated attributes are accurate and stable before they can be used for further analysis. Research in this area was published in [Reinders *et al.*, 1998].

Another objective is to find new attribute calculation methods for important classes of features. This thesis presents an attribute calculation method that describes the shape of features. This part of the research was published in [Reinders *et al.*, 2000].

The feature attributes can be stored in a feature data representation that allows the storage and manipulation of feature attributes. Using this feature data representation, we can investigate the *evolution* of features in time-dependent data.

The next objective is to define correspondence criteria for the comparison of feature attributes and to develop an algorithm to track features in time. This research was published in [Reinders *et al.*, 1999a; Reinders *et al.*, 1999b]. Then, the objective is to define significant events in the lifetime of a feature, and develop algorithms to detect them. This was described in [Reinders *et al.*, 2001].

The final objective is to find ways to visualize temporal patterns in feature evolutions. It must be possible to interactively explore and investigate the evolution in order to obtain a better understanding of the physical phenomena of the application. Results in this part of the research were described in the publications given above.

1.3 Structure of this thesis

The structure of this thesis is as follows.

Chapter 2 provides an overview of related work in the field of feature-based visualization. It defines the conceptual framework used in the rest of this thesis, and describes related work to provide a perspective on the work presented here.

In Chapter 3, 4, and 5 the focus is on attribute representation and calculation. Chapter 3 describes the feature data representation for the storage and manipulation of features and their attributes. Chapter 4 addresses accuracy issues of two commonly used attribute calculation methods: volume integrals and ellipsoidal fittings. Chapter 5 presents an attribute calculation for the topological and geometric description of the shape of a feature, based on the skeleton extracted from a feature object.

Next, the focus of the thesis is directed to the visualization of time-dependent data by means of feature tracking and event detection. Chapter 6 describes the tracking of features based on the feature data representation discussed in Chapter 3. Chapter 7 discusses the detection of events in the evolution of features.

Finally, the techniques described in this thesis are put to the test with several applications in Chapter 8, and Chapter 9 presents some conclusions and gives directions for future research.

Chapter 2

Feature-Based Visualization

In [Post *et al.*, 1999] visualization techniques were grouped in three classes: global techniques, geometric techniques, and feature-based techniques. Global techniques give a qualitative, global visualization of the data at a low level of abstraction. Geometric techniques extract geometric objects (curves, surfaces, solids) from the data. They can be considered as intermediate-level representations, both with regard to locality and level of abstraction. Feature-based techniques extract high-level, abstract entities from the data. The emphasis is on quantification for more precise evaluation and comparison [Silver & Zabusky, 1993].

This thesis is mainly concerned with techniques related to feature-based visualization. This chapter provides a motivation for feature-based visualization, and puts the work described in this thesis in perspective to other research in this field.

This chapter is organized as follows.

Before feature-based visualization is reviewed, Section 2.1 first gives an overview of a number of basic concepts of visualization, which defines a conceptual framework for the rest of this thesis.

Section 2.2 discusses the motivation for feature extraction. Section 2.3 gives an overview of related work in the field of feature extraction. Section 2.5 describes the use of feature extraction for the visualization of time-dependent data, and Section 2.6 gives an overview of related work in this area. Finally, Section 2.7 provides a preview of the research described in this thesis.

2.1 Basic Concepts of Visualization

Visualization transforms data into visual representations. The data may result from numerical simulations or from measurements, i.e. data capture or sensing devices. For simplicity, we will assume that the data is generated from numerical simulations.

2.1.1 The cycle of investigation

Visualization plays an important role in scientific research, especially in the field of computational science. Computational science involves the use of modeling, simulation, visualization, and analysis. Real world phenomena are represented by models which are used in numerical simulations. The numerical simulations result in data sets that describe natural phenomena by means of numbers. These numbers need to be visualized in order to analyze the results of the simulation. The analysis of the data may lead to better models which ultimately leads to a better understanding of the physical phenomena.

Visualization enables the researcher to *observe* the data and explore them: to seek for unexpected phenomena or for confirmation or rejection of a hypothesis. The new insights may lead to better models or to changes in the model parameters, after which the whole process of modeling, simulation, visualization, and analysis is repeated. Thus, the process of investigation is cyclic, and it can be steered using the visualizations.

2.1.2 Data fields and grids

In scientific visualization, data sets represent continuous data fields by samples on discrete points (grid or measure points). The sample points are organized in a *grid* which covers the space of the application domain. The data field between the sample points can be reconstructed by interpolating the data values at these points. Data fields are often three-dimensional (3D), and may consist of multiple data quantities. For instance, a flow data set may include the three components of the flow velocity vector, and in addition scalar quantities such as pressure and temperature.

The grid is used in the numerical simulation; the solution of the numerical simulation is usually computed at the grid points. Hence, the grid define the location of the sample points in 3D space where the data quantities are given.

The grid may be regular, curvilinear, or unstructured (see Figure 2.1). In a regular grid the point locations are regularly distributed over the domain. The grid is structured, which means each grid point can be referenced by integer indices (i, j, k). A curvilinear grid is also structured, but the grid lines (connecting iso-index grid points) are curved. We must have the spatial coordinates for each grid point in order to analyze this type of data. In an unstructured grid, the grid points cannot be referenced by indices, the grid point locations are scattered through space, however connectivity information between points is often available.

The grid spans a volume in 3D space that can be subdivided in small volume elements called *cells*. Usually, the corner points of the cells are the grid points where the data are defined. The cells subdivide the entire application domain, but may not intersect or overlap, and adjacent cells must have common edges

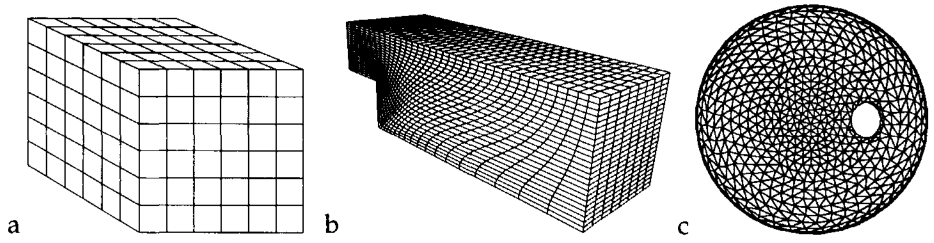


Figure 2.1: Three types of grids: a) regular, b) curvilinear, and c) unstructured.

and faces. The shape of the cells depends on the grid type used. The shape of the cells may be cubes (or voxels) for a regular grid, hexahedra for a curvilinear grid, and tetrahedra for an unstructured grid. But it is also possible to define differently shaped cells with the same set of grid points (for example, each cube can be decomposed into five tetrahedra).

In data from numerical simulations, the specifications of the grid and cells depend on the numerical method used. Numerical methods such as finite-element, finite-difference, or finite-volume methods are based on the definition of a discrete, computational grid that spans the domain investigated. The shape of the grid should be conform to the boundary of the domain. Computational fluid dynamics (CFD) simulations often use curvilinear grids because the boundary of the domain is often curved.

2.2 Motivation for Feature Extraction

Feature-based visualization techniques aim at algorithmic, automated extraction of features from data sets. A feature is any object, pattern or structure in the (visualized) data that is of interest and that is a subject of investigation. The goal is to extract the feature and to calculate quantitative attributes describing the characteristics of that feature. With the attributes, the feature can be visualized and evaluated more precisely. We call *feature extraction* the process of extraction, attribute calculation, and visualization of features.

Feature extraction should give answers to questions such as: 'Does a certain phenomenon occur?', 'When/how/where precisely does it occur?', 'What happens to it when parameters are changed?', and 'How does it develop in the course of time?'. Feature extraction focuses on the occurrence, quantification, and analysis of phenomena. The task of feature extraction is to assist in answering these questions, and to help the user analyze the data and gain new insights in the particular phenomena in these data.

2.2.1 Advantages

Feature extraction has a number of advantages compared to other visualization techniques. The advantages are related to the questions mentioned above about occurrence, quantification, and analysis.

Feature occurrence

The features are explicitly extracted and visualized. The extraction of interesting features is not left solely to the visual perception of the observer. Instead of 'seeing' a feature in the data, the feature is extracted explicitly by an algorithmic, automated procedure. The procedures are based on a definition underlying a feature, and the data are tested to comply to the criteria based on this definition. The procedures are a tool to aid the human perception in the analysis of the data.

The extraction procedure ensures an objective interpretation of the criteria for a feature. Feature extraction may find features that were not visible to the eye, or may reject visible features. In these cases, the scientist may change the criteria until he is satisfied with the resulting features. There is an interaction between the definition for a feature and the criteria that are tested. Often, a working definition for a feature is found after a process of exploration.

The definition of a feature is closely related to its conditions of occurrence, and this is often precisely the purpose of an investigation. Thus, the searching of good extraction criteria can be part of the investigation. As such the process of developing feature-based visualizations is just as important as the final result.

Feature extraction raises the data abstraction to a higher level. A feature is an abstract phenomenon that is more closely related to the concepts of the application than the numbers in the dataset. The focus is redirected from the dataset as a whole to the interesting parts in that data, irrelevant information is ignored. A relatively small part of the data is extracted and characterized. This is a recursive process; it is possible to find features in the extracted features [van Walsum *et al.*, 1996]. At each recursive step, the level of abstraction is raised to a higher level.

Feature extraction often leads to abstract, simplified visualizations giving a schematic representation of complicated phenomena. This is suitable because the features represent the data at a higher level of abstraction. Because the visualizations are simple and schematic, they ensure a fast display and can be visualized interactively.

Feature quantification

Feature extraction provides a quantification of a feature by attribute calculation. Attribute calculation is an important step in the feature extraction since it gives a quantitative measure of the characteristics of the feature. Quantification contributes to the raising in the level of abstraction: from data containing interesting features to data describing the interesting features.

Feature extraction leads to a huge *data reduction* because the characteristics of a feature are usually quantified by a small set of attributes. The transformation from grid data to feature data often yields a reduction in the order of 1000. For instance, a dataset existing of a scalar data field with 128^3 floats has a size of 8 Mb. Suppose the feature extraction finds 50 features (a relatively large number) and quantifies them by ellipsoidal fittings, the resulting feature data file (see Section 3.3) has a size of 9 Kb. The data reduction is in the order of 1000, which is a typical ratio. The applications discussed in Chapter 8 all show similar ratios.

The data reduction obtained by feature extraction could be a solution to the 'tera-byte problem'. With the increasing power of computers the data sets resulting from numerical simulations become larger and larger. Computers are able to generate so much data that it becomes impossible to visualize, let alone analyze every aspect of the data. In the field of visualization this problem of large data sets is referred to as 'the tera-byte problem'.

The data reduction also provides an excellent opportunity for distributed processing. The computationally intensive first part of a numerical simulation and feature extraction can be executed on a remote high performance computer, while the results, the feature data, can easily be transferred to the visualization workstation by a low-bandwidth link. There the features can be visualized in real time.

Feature analysis

The quantification of features leads to possibilities for the comparison and further analysis of features. The attributes can be investigated with respect to simulation parameters or as a function of time.

The way simulation parameters affect the feature attributes may be of interest. For instance, changes in the design of a wing from an aircraft results in different flow patterns (e.g. vortices) above the wing which give the lift to an aircraft needed for flying. The investigation may focus on the optimal design parameters of the wing that results in the best lift. The simulation parameters are changed interactively while running the simulation in the background and viewing the results by visualization. Such systems are called computational steering environments; see [Mulder *et al.*, 1999] for an overview of work in this area.

Another example of further analysis is the investigation of the evolution of feature attributes in time. Interesting *events* or stages in the evolution of the feature can be detected, quantified, and visualized [Silver & Wang, 1996; Reinders *et al.*, 2001]. The events are interesting phenomenon (features) in the evolution of features. Thus the level of abstraction is raised to yet another higher level;

from data describing features to data describing the evolution of features. This shows the recursiveness of feature extraction: one can extract features in the features already extracted, thereby again raising the level of abstraction.

2.2.2 Disadvantages

Next to the number of advantages described above, feature extraction also has a number of disadvantages.

Features are difficult to define, and many application specific feature extraction techniques exist. This by itself is a disadvantage. But additionally, once a suitable extraction method is found, then still questions remain such as 'Is what we found really this feature?', and 'Have we found them all?'. Because of this uncertainty, feature extraction remains a process of exploration. Exploration helps in finding the right questions that may lead to a working definition for a feature.

Feature extraction leads to more abstract visualizations, and therefore may be less intuitive. Often, the context is lost and an explanation is needed because many features do not have concrete shapes and can only be displayed by visual symbols. A convention is needed, e.g. the front lines in Figure 1.1. This in contrast to low-level, global visualizations where the resulting images are self-explanatory. Because feature-based visualizations are abstract, the user often must go back to the original data and create other visualizations that are less abstract, in order to investigate the phenomena. However the advantage is: now he knows where to look and what to look for.

The process is irreversible. It is not possible to go back from the feature attributes to the raw data. It is possible to create a database mechanism for the retrieval of the original grid data. But, these data cannot be derived from the feature attributes since a link between feature characteristics and simulation model parameters is not established. This could be one of the topics for future research.

2.3 Related Work on Feature Extraction

There exist many application specific feature extraction techniques because features are different for each application. This section first describes methods for the extraction of anatomical features, originating from the field of medical visualization. Then, a number of application specific techniques is discussed for the extraction of flow features. Finally, two generic extraction methods are described of which one is the feature extraction technique which was developed in previous work [van Walsum, 1995]. Still, no universal procedure exists to extract all types of features, and therefore exploration remains a vital step in the extraction of features.

2.3.1 Anatomical features

Medical visualization aims at the depiction of the human body for the use in computer-assisted diagnosis and surgical planning [Xu, 1996; Nakajima *et al.*, 1997]. The data are mostly measured by medical devices such as MR and CT scanners, which result in 3D images of the human body. Traditionally, physicians analyze X-ray images visually, and mentally reconstruct the 3D shape of organs and tissue from 2D slices in order to give a diagnosis. Nowadays, user-supervised, automated segmentation and shape reconstruction is becoming widely accepted.

Medical visualization focuses on the anatomy of the human body, it extracts and visualizes organs and tissue. The features that are visualized include the brain, the heart, blood vessels, and specific pathologies such as tumors. The goal is to visualize the geometric structure of the feature. The obtained geometries can be used for virtual walk-through and interactive exploration of the structures [Bartz *et al.*, 1999]. Most of the techniques known in medical visualization focus on the segmentation and surface reconstruction of the anatomical features.

Segmentation

The extraction of anatomical features first requires a segmentation of the images. The segmentation of medical images is complicated because of the complexity and variability of anatomical shapes. Moreover, boundaries between different anatomical structures are often indistinct or even absent because of lack of contrast between these structures, or due to imperfect image quality (noise, sampling artifacts, spatial aliasing, and partial volume effects). This may seriously hamper the accuracy of the segmentation.

Many segmentation techniques from the field of 2D image processing are implemented for medical purposes [Bezdek *et al.*, 1993]. This is possible because often medical data are organized in stacks of 2D slices where the distance between slices is considerably larger then the distance between pixels within a slice. Many segmentation approaches first perform a 2D segmentation within a slice, and then construct a 3D shape by connecting the segmented points in the 2D slices [Leymarie & Levine, 1992b; Atkins & Mackiewich, 1998].

There exists a large body of literature on segmentation of images. Examples are region growing [Adams & Bischof, 1994; Boykov & Jolly, 2000], active contours [Cohen, 1991; McInerney & Terzopoulos, 1996], and snakes [Kass *et al.*, 1988]. Often, the segmentation process is initialized and supervised by the scientist (the physician in this case) by placing control points on the boundary contour of a feature. Within the slices a contour of the feature can be determined, for instance by using the gradient of data values [de Bruin *et al.*, 1999].

Figure 2.2 shows a slice of a medical data set, the control points inserted by the physician, and the contour that is found. The figure shows two segmented areas, each indicating a different human organ. After extending the contours to subsequent slices, the result of the segmentation step is a labeled voxel grid where each label indicates the anatomical structure the voxel belongs to.

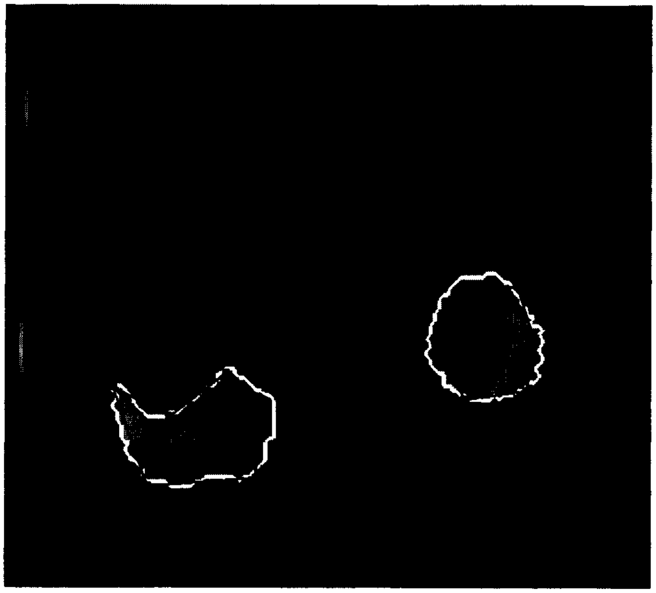


Figure 2.2: Segmentation of medical features by a gradient contour finding technique, [de Bruin *et al.*, 1998]. The control points added by the physician are shown in black and the resulting contours are shown in white.

Surface reconstruction

The surface of the anatomical structure can be reconstructed by a triangular mesh or by fitting some anatomical model over the segmented areas. Using the triangular mesh or the model parameters, we can create a 3D geometry that can be displayed using 3D computer graphics.

Generating a triangular mesh over the surface of an object, e.g. by marching cubes [Lorensen & Cline, 1987], may result in millions of triangles that are hard to display interactively and that often contain many unwanted artifacts. The number of triangles can be reduced while still preserving the characteristic features of the surface by surface simplification [Schroeder *et al.*, 1992; Klein *et al.*, 1996; Cignoni *et al.*, 1998; Frank & Lang, 1998]. The process of surface simplification is particularly problematic because the distance between slices is much larger then the distance between pixels within a slice. Artifacts such as staircases may result. Surface nets [Gibson, 1998] is one way to provide a solution to this problem.

Another approach for the reconstruction of shape is the use of anatomical models. The features can be modeled by finding a 'best fit' between a parametric model and the image volume [Lelieveldt, 1999]. The model can then be visualized by mapping the model parameters onto a parametric geometric object that can be displayed. For instance, the vascular structure can be modeled by a tree-structure of generalized cylinders [Puig *et al.*, 2000], after which the blood vessel can be visualized by sweeping a contour cross-section along the model trajectory. With this visualization we can easily recognize vascular pathologies such as aneurysms and stenoses.

2.3.2 Flow features

In the field of computational fluid dynamics (CFD) many techniques exist for the extraction of particular flow features. Examples of flow features are vortices, critical points, shock waves, separation lines and surfaces, re-circulation zones, and transition to turbulent flow. An overview of flow feature detection techniques can be found in [Kenwright, 1999]. Here, a number of specific extraction techniques are reviewed. They seem unrelated, but [Peikert & Roth, 1999] showed that a number of these specific techniques can be captured by a common principle, the 'parallel vectors' operator.

Critical points and flow topology

Critical points are points in a velocity field where all three vector components of the velocity are zero, i.e. $\mathbf{v} = \mathbf{0}$. These points can be characterized according to the behavior of nearby tangent curves. If we consider the Taylor series expansion of a 2D field in the neighborhood of such a point, then the first order partial derivatives of the field determine the vector field's local behavior. Thus, for a non-degenerate critical point (x_0, y_0) we can use the matrix of these derivatives, i.e. the Jacobian matrix, to characterize the vector field and the behavior of nearby tangent curves:

$$\left[\frac{\partial(u,v)}{\partial(x,y)}\right]_{x_0,y_0} = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}_{x_0,y_0} \tag{2.1}$$

The eigenvalues and eigenvectors of this matrix are of particular interest because they provide a classification of the critical points [Helman & Hesselink, 1989; Helman & Hesselink, 1991]. The eigenvalues are complex and are used to classify the critical points, see Figure 2.3. The critical point can be an *attracting node*, a *repelling node*, an *attracting focus*, a *repelling focus*, a *center*, or a *saddle*. In this classification, the center indicate the center of a vortex structure that was discussed in the previous paragraphs.

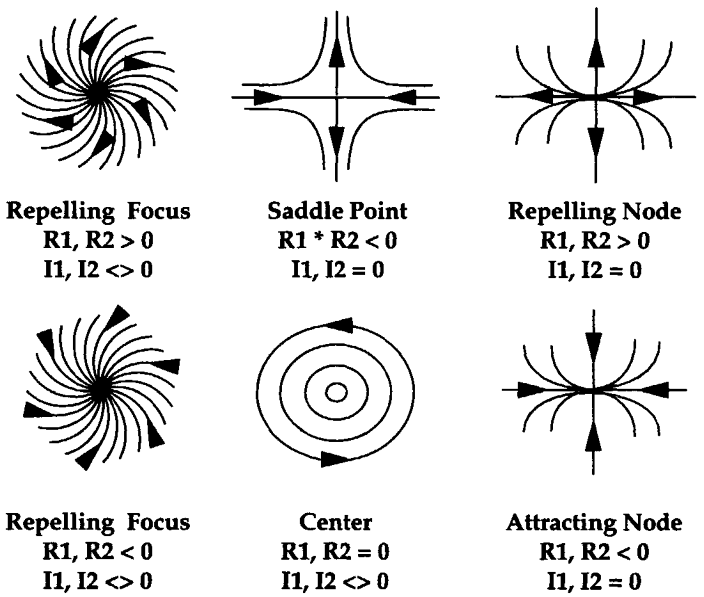


Figure 2.3: Classification criteria for critical points. R1 and R2 denote the real parts of the eigenvalues of the Jacobian, I1 and I2 the imaginary parts, [Helman & Hesselink, 1989].

Other references to publications in the visualization literature about critical points are [Lavin *et al.*, 1997; Scheuermann *et al.*, 1997; Lavin *et al.*, 1998; Batra & Hesselink, 1999].

The *flow topology* can be analyzed using the critical points, and the separation and attachment lines discussed in the next paragraph. The characteristic points can be linked into a partially connected graph representing the topology. The curves connecting the various critical points can be obtained by integrating forward and backward starting from the critical points along the incoming and outgoing directions of the eigenvectors. The resulting schematic graph representation helps us to investigate flow topology.

Figure 2.4 shows the flow topology generated for a computed 2D flow around a circular cylinder (Figure 2 of [Helman & Hesselink, 1991]). The flow is incident from the left, with one instantaneous streamline ending directly on the front of the cylinder. Points *de*, *at*, *ce*, and *sp* denote detachment, attachment, center, and saddle points, respectively. All instantaneous streamlines starting above that curve are deflected over the top of the cylinder, and those starting below are deflected beneath it. Vortex shedding occurs behind the cylinder, as indicated by the detachment-attachment 'bubble' in the topology, which develops into a paired saddle and center later in time.

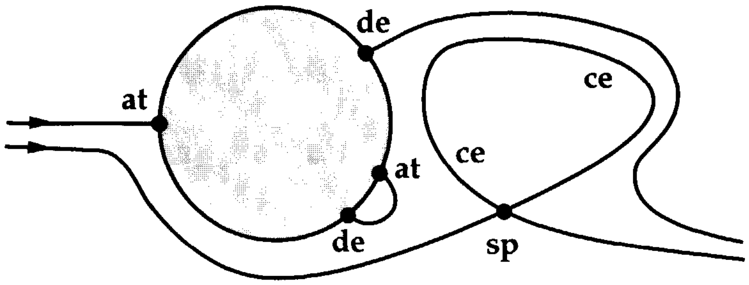


Figure 2.4: Schematic representation of flow topology of a flow around a circular cylinder, [Helman & Hesselink, 1991].

Very complex topologies can be present in turbulent flow, leading to visualizations with a cluttered image which are difficult to interpret. In [de Leeuw & van Liere, 1999], the visualization is simplified by collapsing pairs of critical points while still preserving a consistent topological map. By only displaying the most important critical points, a simplified topological map can be obtained. Other publications about flow topology are [Helman & Hesselink, 1990; Globus *et al.*, 1991; Delmarcelle & Hesselink, 1992; Lavin *et al.*, 1997].

Separation and attachment

In addition to critical points, certain points and lines on the walls of objects or bodies in a fluid flow can be important: the separation and attachment points and lines. Flow separation and attachment occur when a flow abruptly moves away from or returns to the solid body such as the surface of an aircraft. The lines along which this occurs are called separation and attachment lines. Because we often associate separation lines with vortices and recirculation zones, determining separation topologies is important both for understanding fundamental fluid dynamics and practical applications in aircraft and nozzle design.

In [Kenwright, 1998], a method is described that extracts separation and attachment lines automatically using a *phase plane* analysis. Some other publications about separation and attachment lines include [Helman & Hesselink, 1990; Globus *et al.*, 1991; Pagendarm & Walter, 1994; Peikert & Roth, 1999].

Shock waves

Supersonic and hypersonic flows often show discontinuities of some physical quantity, such as pressure, known as shock waves. Determining the exact location and structure of shock waves in computed flow solutions is not straightforward. Though physical shocks are very sharp, numerically computed shocks are ordinarily smeared over several grid cells, due to limited resolution of the computational grid. Moreover, the grid point locations rarely coincide with the shock location, so that interpolation issues arise.

In [Pagendarm & Seitz, 1993] shock waves are extracted by determining zero crossings for the second derivative of the desired quantity. Iso-surfaces are created by a threshold at the maximum or minimum of the pressure gradient in the direction of the flow velocity. The results are flat surface-like features that represent the shock waves. Other references to publications about the detection and visualization of shock waves are [Ma *et al.*, 1996; Lovely & Haimes, 1999].

Vortices

Vortices are well known features in the field of CFD. However, it is surprisingly difficult to establish a definition of a vortex that is robust enough to locate all the coherent structures that a flow physicist would consider to be vortices. One working definition is [Robinson, 1991]:

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

With this definition in mind a number of methods were developed for extracting vortex cores and for finding spiral streamline patterns.

There are many publications about vortex detection: [Sujudi & Haimes, 1995; Roth & Peikert, 1996; Kenwright & Haimes, 1997; Portela, 1997; Haimes & Kenwright, 1999]. Here, we will give two examples that provide nice intuitive solutions to the detection of a vortex.

In [Banks & Singer, 1995] a predictor-corrector algorithm is described that extracts the vortex core. The method is illustrated in the schematic diagrams of Figure 2.5. Starting from a user-specified seed point, the next position of the vortex core is predicted by integrating along the vorticity vector $\omega = \nabla \times \mathbf{v}$ (top of the figure). The predicted point typically misses the vortex core. Next, the heuristic is used that the centripetal acceleration within a vortex results in a local pressure minimum at the core. In a plane perpendicular to the core, the pressure minimum is expected to coincide with the point where the core pierces the plane. The predicted point is corrected to the pressure minimum in a plane that is perpendicular to the vorticity vector (bottom of the figure). Hence, a vorticity-predictor, pressure-corrector method is used for finding the vortex core.

The next step is to construct a tube around the vortex core. The cross-section is created by sampling the vorticity along radial lines emanating from the core

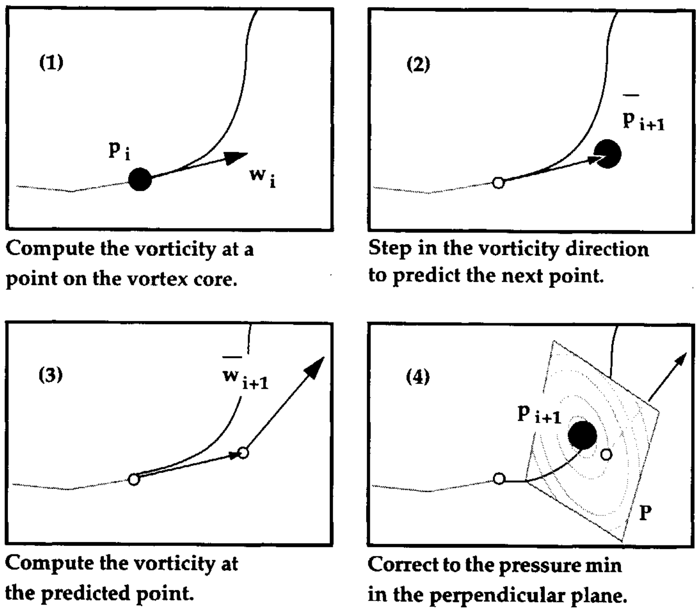

(1)
P i
w i
Compute the vorticity at a point on the vortex core.
(2)
P i+1
Step in the vorticity direction to predict the next point.
(3)
w i+1
Compute the vorticity at the predicted point.
(4)
P i+1
P
Correct to the pressure min in the perpendicular plane.

Figure 2.5: Four steps of the predictor-corrector algorithm for extracting the vortex core, [Banks & Singer, 1995].

in a plane perpendicular to the core vorticity. Steps are taken along the line until local vorticity violates a certain threshold condition. Using the cross-sections, the vortex tube geometry is reconstructed.

The second example of vortex extraction uses the *winding angle* of streamlines [Portela, 1997; Sadarjoen & Post, 1999]. Streamlines are calculated in a 2D plane which should be taken perpendicular to the (expected) vortex core. Looping streamlines are selected from all streamlines by testing the winding angle. The winding angle is the sum of the angles between the line segments of a streamline (see Figure 2.6). The selection criteria for the streamlines are: the winding angle $|\alpha_{w,i}| \geq 2\pi$, and the distance between the starting point and the final point is relatively small compared to the average radius. The latter criterion is used to ensure locality.

After the selection of streamlines, the selected streamlines are clustered in order to group the streamlines belonging to the same vortex. The clustering is based on the center point or the geometric mean of the streamline positions $\mathbf{P}_j$. Once clustered, the streamlines in each cluster are used to quantify the vortex,

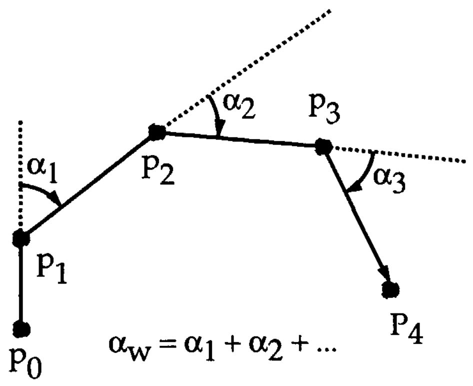


Figure 2.6: The winding angle $\alpha_{w,i}$ is the sum of the angles between the edges, [Sadarjoen & Post, 1999].

i.e. to calculate quantitative attributes. The shape of the vortices is approximated by 2D ellipses (see also Section 4.1.2). In addition, a number of specific vortex attributes can be calculated: the vortex rotational direction, and vortex angular velocity:

streamline center:	$\mathbf{S}_i = \frac{1}{\lvert S_i\rvert} \sum_{j=1}^{\lvert S_i\rvert} \mathbf{P}_{i,j}$
cluster center:	$\mathbf{C}_k = \frac{1}{\lvert C_k\rvert} \sum_{l=1}^{\lvert C_k\rvert} (\mathbf{S}_{k,l})$
cluster covariance:	$M_k = \text{cov}\,(\Psi(\mathbf{C}_k))$
ellipse axis lengths:	$\lambda_k = \text{eig}\,(M_k)$
ellipse axis directions:	$\mathbf{d}_k = \text{eigvec}\,(M_k)$
vortex rotational direction:	$d_k = \text{sign}\,(\alpha_{w,k})$
vortex angular velocity:	$\omega_k = \frac{1}{\lvert C_k\rvert \Delta t} \sum_{l=1}^{\lvert C_k\rvert} \alpha_{w,l}$

where $|S_i|$ is the number of points in streamline S_i, $C_k = \{S_{k,1}, S_{k,2}, \cdots\}$ is a cluster of streamlines, $S_{k,l}$ is streamline #l in cluster #k, $|C_k|$ is the number of streamlines in that cluster, and $\Psi\,(C_k)$ are all the points on all the streamlines in cluster k.

Figure 2.7 shows two vortices extracted with the winding angle method. The vortices appear in the wake of a tapered cylinder. This data set and the utilization of the winding angle method are reviewed in more detail in the application described in Section 8.4.2.

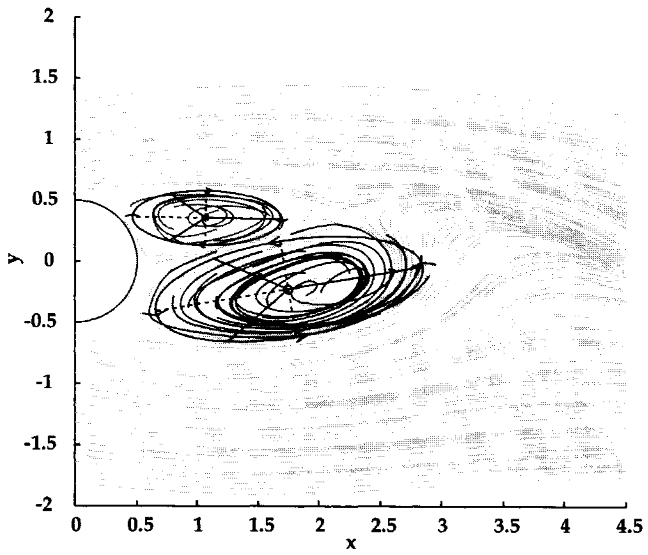


Figure 2.7: Vortices detected by the winding angle method in the wake of a tapered cylinder, [Sadarjoen & Post, 1999].

2.3.3 Generic feature extraction techniques

The work discussed so far describes techniques to extract specific features. Also, a number of generic feature extraction techniques exist. Two of them are: selective and iconic visualization, and linked derived spaces.

Selective and iconic visualization

Selective visualization [van Walsum, 1995] is based on the principle that features are identifiable by local regions of interest where certain data values are within a specific range of values. Selective visualization uses a *selection expression* that identifies the grid points where the data values are within a user-specified range. The selection expression may include derived quantities from the data and combinations of multiple threshold values. This expression yields TRUE or FALSE for every grid point in the data. Hence, it results in a number of selected grid points that usually form connected clusters or regions of interest.

For each region of interest a number of attributes is calculated, describing the characteristics of the features. The attributes may describe the feature's geometry, data, or combined data-geometry properties. The resulting attributes can be mapped onto the parameters of a parameterized iconic object [van Walsum *et al.*, 1996]. Thus, a selective and iconic visualization is obtained.

The feature extraction step used in this thesis is usually based on selective and iconic visualization. Therefore, this process is discussed in more detail in Section 2.4.

Linked derived spaces

Feature detection in *linked derived spaces* was introduced by [Henze, 1998]. A linked derived space consists of a data browser in which many graphs of 'coordinate spaces' can be displayed simultaneously. The coordinate spaces are xy-plots where two data values are plotted against each other, while still preserving the original mesh connectivity. For instance, Figure 2.8a shows the original mesh in the physical (x, y) space and 2.8b shows the derived space of v_x versus v_y (with (v_x, v_y) the two velocity components).

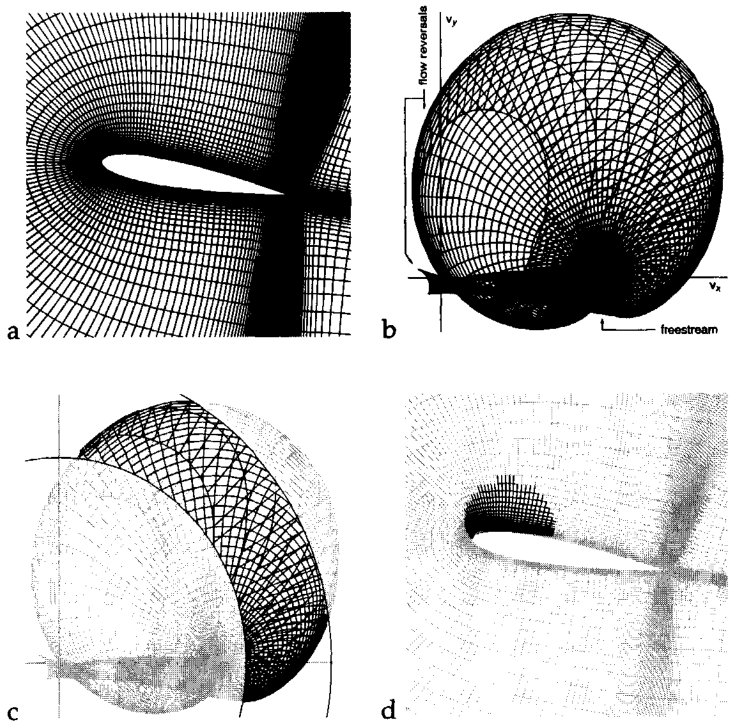


Figure 2.8: Linked derived spaces of an airfoil dataset, [Henze, 1998]: a) in physical space, and b) in velocity component space. Selections: c) in velocity component space result in d) a corresponding image in physical space.

The field data in the derived spaces retain the original mesh connectivity, although it is redistributed into new geometric patterns in each coordinate space. The various coordinate systems preserve the spatial connectivity and temporal variability. The coordinate axes are specified by some subset of the dependent variables (or quantities derived from them). The position in the derived space by itself indicates the dependent variable value.

The linked derived spaces can be used to explore data characteristics, in search of data points with interesting data. For instance, the free-stream flow from left to right in physical space, Figure 2.8a, is a point on the positive x-axis in the derived space, Figure 2.8b. Also, the grid points on the left of the y-axis in the derived space are data points with negative v_x, and hence indicate areas with flow reversal.

Selections can be made with a brush tool in one image, resulting in linked image selections in the other images, as can be seen in Figure 2.8 c and d. The different images in derived spaces are linked together. The corresponding selections in the linked images are calculated and displayed.

The selections in the linked derived spaces are very similar to the selection expressions used in selective visualization. In fact, any selection made in the linked derived spaces can be transformed to a selection expression. Linked derived spaces also allows combined multiple selections with boolean operators. The functionality of the linked derived spaces is the exploration of parameter spaces, e.g. for the use of feature detection.

2.4 Feature Extraction Pipeline

The selective and iconic visualization process mentioned above (Section 2.3.3) was developed at the Computer Graphics group at Delft University of Technology [van Walsum, 1995]. It can be modeled by the pipeline shown in Figure 2.9. The pipeline consists of the following steps: selection, clustering, attribute calculation, and iconic mapping.

The process of feature extraction is controlled by the scientist in the sense that his knowledge of the data and his conceptual model of an interesting feature are translated into several options and parameters used during the extraction process: the selection expression, the connectivity criteria, the calculation method and the mapping function. These will be explained below.

Selection

Selection identifies the grid points where the data satisfy the criteria of what is interesting. The data in each grid point are evaluated by a *selection expression* that yields TRUE or FALSE. The general idea is that scalar quantities are tested to one or more thresholds in order to select a range of data values. The criteria

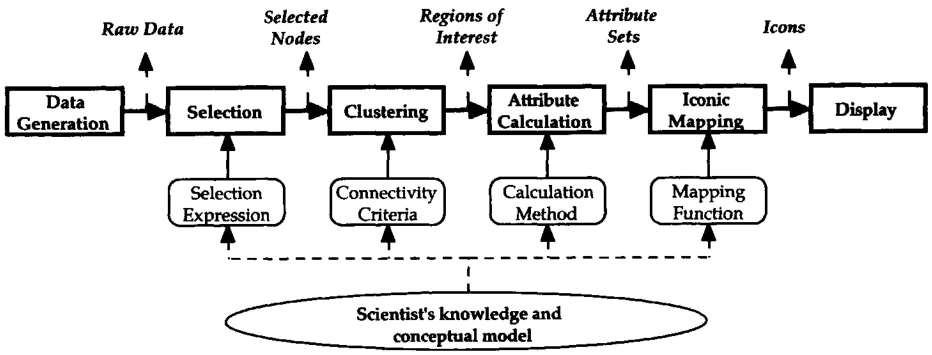


Figure 2.9: The feature extraction pipeline based on the selective and iconic visualization process.

for 'interesting' is expressed by a logical combination of threshold values testing the raw data and/or derived data quantities; it is possible to use multiple thresholds.

The selection expression is a mathematical formulation of the underlying physics of the feature. For instance, a vortex can be defined as the region where vorticity is high and pressure is low:

$$S_{i,j,k} = \|\nabla \times \mathbf{v}_{i,j,k}\| \geq T_v \wedge P_{i,j,k} \leq T_p \tag{2.2}$$

where $S_{i,j,k}$ is the selection result, $\mathbf{v}_{i,j,k}$ is the velocity vector at the grid point with indices (i, j, k), $P_{i,j,k}$ is the pressure at that point, and T_v and T_p are threshold values for the vorticity and pressure respectively.

A language was developed with a set of operators (boolean, scalar, vector, and matrix operators) and functions (gradient, and statistical functions) in order to specify the selection expression. With this language almost any selection can be created based on the data values, and on the point positions. For instance, equation 2.2 can be expressed as (the threshold values $T_1 = 0.8$ and $T_2 = 0.2$ can be found by exploration):

```
; calculate the vorticity 'vort'
vort = curl(velocity)
; calculate the magnitude of the vorticity 'mag_vort'
mag_vort = len(vort)
; set the two threshold values
T1 = 0.8
T2 = 0.2
; create the selection 'select_out'
select_out = mag_vort > T1 AND pressure < T2
```

It shows that data quantities can be derived from the original raw data, and that the selection can be based on a combination of multiple thresholds.

The evaluation of a selection expression yields TRUE or FALSE, whether points are selected or not. The result of the selection step is a binary data field. The selection step can also be replaced by any segmentation technique which produces a binary field as output. In the field of image processing many segmentation algorithms exist, e.g. see Section 2.3.1.

Clustering

The clustering step gathers neighboring selected points into regions of interest. Individual selected points have no other meaning than an indication of the positions where the data satisfy the criteria of interest. However, our concern is finding coherent regions consisting of more than one point: regions of interest. Therefore, the task is to find clusters of connected points that are part of the selection.

The clustering procedure has two connectivity criteria: the connected component definition and the cluster threshold. The connected component definition determines the adjacency of points, whether they have 6 neighbors (face-connected), 18 neighbors (edge-connected), or 26 neighbors (vertex-connected). The cluster threshold is equal to the minimal number of points needed to form a cluster. Clusters of just a few points may be the result of noise or inaccuracy, and can be eliminated using the cluster threshold.

Each cluster represents a region of interest in the data, and is considered as a feature. Clustering assigns a label to every selected point that identifies the cluster number, and the feature identity.

Attribute calculation

Once the clusters have been identified, each cluster is used to calculate attributes that characterize the feature. There are many possible methods to calculate attributes. Methods can be applied from areas such as computer vision (object fitting methods) and image processing (morphological operators, skeletonization). The user can add any attribute that seems interesting for this type of feature. More than one calculation method can be used, each method resulting in an additional set of attributes.

Chapter 4 presents two attribute calculation methods: the volume integral and the ellipsoid fitting. Volume integrals are used to calculate global integral attributes of a cluster. Examples are the center position of the cluster points, and the volume of the cells around the points. These are basic attributes describing global characteristics of a feature.

Ellipsoid fitting determines the 'best fit' of an ellipsoid around the cluster of points. They can be calculated using the variance/covariance matrix of the point positions [Fung, 1965], which can be calculated by a volume integral. The ellipsoid has the following parameters: the center position, the axes lengths, and the axes orientations. In 3D, this result in a set of nine attributes (nine degrees of freedom). Ellipsoid fittings are commonly used and provide a good sense of position, and size.

The representation of a feature by a set of attributes leads to a large data reduction. We must carefully select the attributes to be stored, because information is lost. The attributes should describe the feature as precisely as required for the purpose. Therefore, we need a wide range of methods for attribute calculation in order to obtain all relevant characteristics of a feature.

Iconic mapping

Once the attributes have been determined, the characteristics of the features can be clearly visualized. An iso-surface can be generated using the segmented data, but attributes of the feature can be visualized more directly by mapping them onto the parameters of a parametric icon or glyph [van Walsum *et al.*, 1996].

An icon is a symbolic object that gives a visual representation of the features, and which can relate to the physical concepts and visual languages of the area of application. An icon can be represented as a geometric object with a parametric shape and visual attributes that can be linked to the attributes of a feature. The relation between the parameters of the icon and the attributes is called the mapping function. The goal of iconic mapping is to visualize essential characteristics of a feature in a symbolic representation.

As with the number of features, the number of different types of icons is unlimited; the design of an icon depends on the specific application area and research problem. In [van Walsum *et al.*, 1996] a icon modeling language is introduced that can create a wide range of icons.

A fast display is guaranteed since the icons are relatively simple geometric objects with a limited number of triangles. Figure 2.10 shows a number of icons that can be used to display ellipsoid fitting attributes.

2.5 Visualization of Time-Dependent Data

It is feasible to solve time-dependent equations with numerical methods, resulting in time-dependent data sets. Time-dependent data constitute of data sets for each time step (frame) defined on a grid that may also change in time. Thus, thousands of frames are calculated each resulting in a dataset that can be visualized and explored. The amount of data is so huge that it is becoming difficult to store it, let alone to visualize and analyze it. Examples of large time-dependent data sets are listed in Table 2.1, from [Kenwright, 1999].

Visualization of large data sets, such as time-dependent data, is difficult because the data cannot be held in main memory. Furthermore, the exploration is

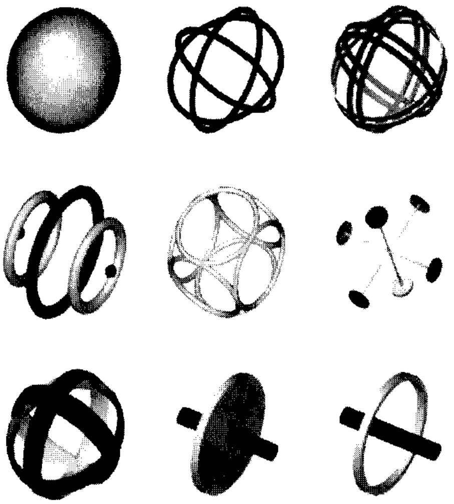


Figure 2.10: Different icons to visualize ellipsoids.

Data set name and year		# vertices	# time steps	size (Mb)
Tapered Cylinder	'90	131,000	400	1,050
McDonnell Douglas F/A-18	'92	1,200,000	400	12,800
Descending Delta Wing	'93	900,000	1,800	64,800
Bell-Boeing V22 tiltrotor	'93	1,300,000	1,450	140,000
Bell-Boeing V22 tiltrotor	'98	10,000,000	1,450	600,000

Table 2.1: List of time-dependent data sets, from [Kenwright, 1999].

often very tedious because the user cannot change visualization parameters interactively without waiting a long time for the result. This is caused by the fact that data processing consumes much time, and the geometric objects generated by the visualization are often very large (with many triangles).

One way to visualize time-dependent data is to create animations off-line. It is possible to create global visualizations of each frame and save these as 2D images, or 3D scenes. Animation of the images or scenes provide a dynamic impression of the phenomena in the time-dependent data. Additionally, the visualization technique can be adapted in order to emphasize the dynamics [Becker & Rumpf, 1998].

Off-line creation of animations has the deficiency that it does not allow interactive manipulation of visualization parameters. In case of 2D movies, there is an additional disadvantage: the viewpoint is set, so that the only interaction is provided by the video-functions. However, the advantage of 2D movies is that display frame rate only depends on image size, not on scene complexity.

When viewing an animation, the user visually scans for coherently moving structures, and he tries to track apparent features. Playback animation leaves the extraction of these features to the visual perception of the observer, and it does not give a quantitative description of the evolving phenomena. These are all reasons to use a feature-based approach.

Another way to process and visualize time-dependent data is to investigate the evolution of features. The feature-based approach for the visualization of time-dependent data can be achieved in four steps (Figure 2.11): feature extraction, feature tracking, event detection, and visualization.

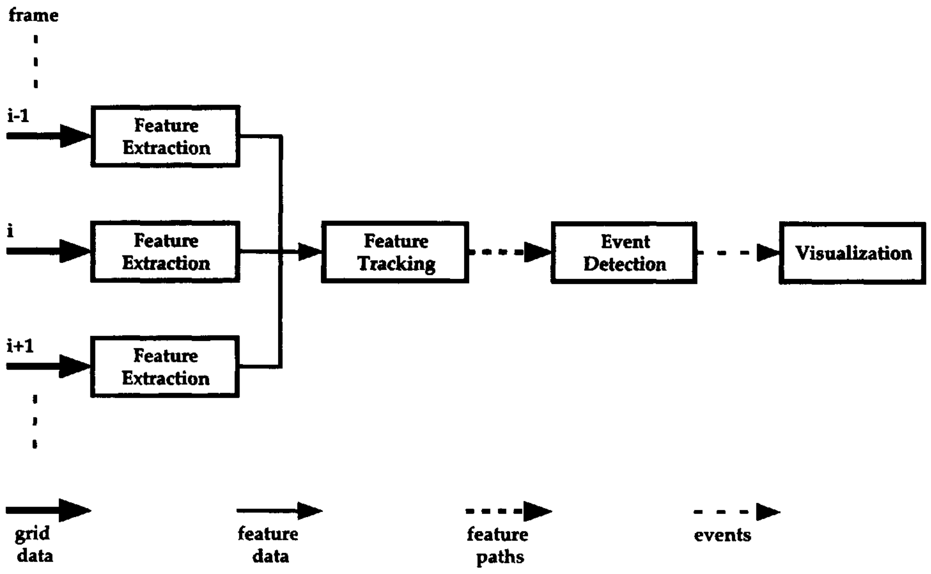


Figure 2.11: The feature-based approach for the visualization of time-dependent data.

1. **Feature extraction**. Extract the features from each frame in a preprocessing step, calculate and store their attributes. A *correspondence problem* [Ballard & Brown, 1982] remains; it must be determined which feature in frame i corresponds to which feature in frame $i + 1$. This correspondence problem can be solved by tracking the features from one frame to the next.

2. **Feature tracking**. Solve the correspondence problem based on the attribute descriptions of features in successive frames. Feature tracking results in path descriptions of continuously evolving features. The next step is to search for interesting evolving phenomena, or events.

3. **Event detection.** Detect certain interesting events by investigating the evolution of features. An event is any significant change in the life cycle of a feature, or a change of state of a feature that is interesting and worthwhile to investigate. Examples of events are birth, death, split, merge, collision, transition to another type of feature, or unusual changes. Each event has its own specific criteria that should be met.

4. **Visualization.** Interactively visualize the evolution of features in a player, show the relations between features in successive frames, and highlight particular events.

The feature-based approach results in entirely new ways of visualizing evolving phenomena. It allows us to select one feature, create a description of its lifespan, describing its motion, growth, interaction with other features, and to visualize that evolution. Also, the time-series as a whole can be investigated in search of occurring patterns in the evolution of features.

2.6 Related Work on Feature Tracking

The correspondence problem is an issue in several scientific disciplines such as image processing, computer vision, and scientific visualization. Although each discipline has its own view to the problem, many approaches are related. The methods can be classified in two categories: 1) pixel-based methods, and 2) feature-based methods.

2.6.1 Pixel-based methods

In pixel-based methods, the correspondence is determined on a pixel-to-pixel basis. There is no separate object recognition step, the correspondence is determined directly based on pixel patterns. Each pixel of the image in one frame is compared to all, or part of, the pixels of the image in the next frame.

A well known technique to determine the correspondence between two images is to calculate the cross-correlation [Castleman, 1996]:

$$R_{fg}(\tau) = f(t) \star g(-t) = \int_{-\infty}^{\infty} f(t)g(t+\tau)dt \tag{2.3}$$

In a sense, the cross-correlation function indicates the relative degree to which two functions agree for various amounts of miss-alignments (shifting). The shift that results in the maximal cross-correlation between two images (or parts of two images) is related to the motion of objects in the images. Maximization of the cross-correlation function between a *target* area from the first image and a *search* area from the second image, is a technique that is often used in practice. Especially, in astrophysics a number of algorithms exist that are based on the maximization of the cross-correlation [Rossow *et al.*, 1980; Rossow *et al.*, 1990; Toigo *et al.*, 1994].

Another approach is the use of optical flow [Hildreth, 1984; Adiv, 1985]. In computer vision, the motion of objects is determined by calculating the optical flow. For each pixel a displacement vector is estimated, which results in a field of vectors: the displacement field or optical flow. Let $f(x, y, k)$ be the image intensity of the point (x, y) at time k, then the displacement components Δx and Δy can be estimated by:

$$\Delta f(x,y,k+1) \approx -\Delta x \frac{\delta f(x,y,k)}{\delta x} - \Delta y \frac{\delta f(x,y,k)}{\delta y} \tag{2.4}$$

For each pixel we have two unknowns (Δx and Δy) but only one equation. The problem can be solved by applying a reasonable additional constraint [Hildreth, 1984]. Examples of such a constraint are based on assumptions such as: each pixel and its neighboring pixels are moving with the same velocity; the displacements are consistent with a rigid motion on the image plane; the velocity field is smooth.

The pixel-based approaches are computationally expensive because they require an optimization process. The optimization is still feasible for 2D images, but the number of calculations becomes immense for 3D data fields, especially when vector fields are concerned. For data fields the feature-based methods are much more suited.

2.6.2 Feature-based methods

The feature-based methods first extract significant features, such as points, lines, contours, or volumes. Then the correspondence problem is solved by comparing the extracted features in successive frames. The comparison of features can be achieved by two mechanisms: region correspondence and attribute correspondence.

Region correspondence

Region correspondence compares the regions of interest that were obtained by extraction or segmentation. The input consists of two binary images containing the different features at each time step. Region correspondence compares these binary images in order to solve the correspondence problem.

The objects in successive binary images can be corresponded by similar methods as the pixel-based methods, only performed on the objects. The objects are used to limit the search area. Correspondences can be established using a minimum distance or a maximum cross-correlation criterion [in den Haak *et al.*, 1992]. Also, it is possible to minimize an affine transformation matrix [Kalivas & Sawchuk, 1991].

Another way of region correspondence is the extraction of iso-valued features in 4D scalar data, where time is considered as the fourth dimension [Weigle & Banks, 1998]. The complete series of time-steps is considered as one data set, in which an iso-surface is determined. Hence, the correspondence between 3D objects is entirely based on spatial overlap. Two objects in successive frames correspond when they share one selected point in space. This method is simple, however it fails when objects in successive frames accidentally overlap but do not correspond, or if a small object moves fast and does not overlap in successive frames.

Spatial overlap between features in successive frames is a simple criterion for correspondence. The problem of accidental overlap can be solved by additional tests on the attributes of the two features. [Silver & Wang, 1996] introduced a tracking method based on a combination of overlap and attribute correspondence. Features are represented by a representation of the segmented volume (an octree representation in the earlier work, and a linked-list representation in later work) in combination with a number of attributes such as position and size. Spatial overlap between features in successive frames is established by testing the leaves of the octree, after which correspondence is established by testing the attributes in a similar fashion as in [Samtaney *et al.*, 1993] (discussed in the next subsection).

The spatial overlap limits the number of features that must be compared, for that purpose it is also useful in the detection of certain events. [Silver & Wang, 1996] use the spatial overlap as a neighborhood criterion in order to limit the number of combinations that is tested for split/merge events. A split is detected by testing the attributes of every combination of all the features in the next frame that overlap with the feature in the current frame. Since the number of features that overlap is limited, the number of combinations is limited. Additional work of this tracking approach is presented in [Silver & Wang, 1997; Silver & Wang, 1998; Silver & Wang, 1999].

The region-based correspondence methods are computationally less expensive than the pixel-based methods since they work on part of the data, i.e. on the features. However, they still need a representation for the segmented data, and therefore are memory consuming. Also, the overlap criterion fails in tracking small features that move fast.

Attribute correspondence

Attribute correspondence uses the calculated attributes in order to compare features. Examples of attributes are the center point position, the volume, the mass, or average data values. The attributes describe the characteristics of the features, and do not need the original grid data. Usually, the attributes are a small set of numbers that consume little memory. It is possible to store the feature attributes of the whole time series in main memory. This is a significant advantage, since now it is possible to track forward and backward through the frames in multiple passes without losing interaction. The user can explore the tracking process by changing tracking parameters and viewing the results.

Since we use a feature extraction technique that results in feature attributes, it is obvious to track features using attribute correspondence. Examples from the literature that use attribute correspondence are the tracking of markers [Sethi & Jain, 1987; Salari & Sethi, 1990; Sethi *et al.*, 1994], the tracking of clouds [Arnaud *et al.*, 1992], and the tracking of flow features [Samtaney *et al.*, 1993; Samtaney *et al.*, 1994].

Below we will discuss two of these methods in more detail: Sethi's greedy exchange method [Sethi *et al.*, 1994] and Samtaney's feature tracking method [Samtaney *et al.*, 1994].

Sethi's greedy exchange method

In [Sethi *et al.*, 1994], markers or *tokens* in images are tracked for vision tasks such as object tracking, motion capture, and structure from motion. The basic property (attribute) of each token is its position, but it is also possible to include other properties (such as volume and mass). The correspondence problem is solved by establishing trajectories in the *property space*, i.e. the attribute space. The algorithm is based on path coherence and smoothness of motion. The total property coherence for all trajectories is optimized.

The basic approach for optimization is to get initial trajectories by connecting the closest (in property space) tokens and then iteratively refine them in a systematic fashion till a terminating condition is met. The iterative refinement consists of exchanging tokens in paths and calculating the gain of the exchange, the path with maximum gain is chosen. Tokens are only exchanged if they lie within a certain distance maximum d_{max} in the property space, thus not every possible connection is tested. The method is referred to as the greedy exchange (GE) method. It is also possible to use another optimization method based on simulated annealing (SA).

The gain of each exchange is calculated using a property coherence measure for three successive mappings of a feature. This property coherence measure is given by:

$$\begin{aligned} F(a,b,c) &= w_1\left(1-\frac{\overline{ab}*\overline{bc}}{\|\overline{ab}\|*\|\overline{bc}\|}\right)+ \\ & \quad w_2\left(1-2\frac{\sqrt{\|\overline{ab}\|*\|\overline{bc}\|}}{\|\overline{ab}\|+\|\overline{bc}\|}\right) \end{aligned} \tag{2.5}$$

where *a*, *b*, and *c* respectively represent the three feature mappings in a k-dimensional property space. The first term on the right hand side gives a measure of change in the direction of the movement (*directional coherence*), and the second term provides a measure of the speed of the movement (*speed coherence*). Both of these measures are combined through suitable weights w_1 and w_2.

For example, the 2D property space in Figure 2.12 consists of the (x, y) positions of three successive features. The first term in equation 2.5 measures the angle between vectors $b - a$ and $c - b$, and the second term measures the difference in length between vectors $b - a$ and $c - b$. In this case, only the (x, y) coordinates are taken into account, but it is also possible to calculate a property coherence measure for additional attributes, when available.

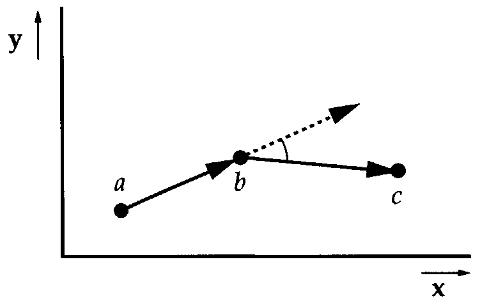


Figure 2.12: The property space of three feature mappings *a*, *b*, and *c*.

Also, the property coherence measure is provided with suitable scaling terms for each axis of the property space:

$$\begin{aligned}\overline{ab} * \overline{bc} &= \sum_{i=1}^{k} s_i(r_{bi} - r_{ai})(r_{ci} - r_{bi}) \qquad (2.6)\\ \|\overline{ab}\| &= \sqrt{\sum_{i=1}^{k} s_i(r_{bi} - r_{ai})^2}\\ \|\overline{bc}\| &= \sqrt{\sum_{i=1}^{k} s_i(r_{ci} - r_{bi})^2}\end{aligned}$$

where s_i is the scaling factor for the i^{th} axis of the property space and $\sum s_i = 1.0$, and r_{ai} is the i^{th} property value of feature a. With these scaling terms, one axis can be assigned more weight than another.

Samtaney's feature tracking method

[Samtaney *et al.*, 1994] introduced the following evolutionary events (see Figure 2.13): continuation, creation, dissipation, bifurcation, and amalgamation. First, all continuations are determined by testing each object in frame i to the nearest (in position) object in frame $i+1$. Then, combinations of the remaining unmatched objects are tested for bifurcation and amalgamation. Finally, all objects that could not be matched in one direction are classified as either dissipated or created. Thus, features are linked on a two-frame basis resulting in the paths and evolutions of individual objects.

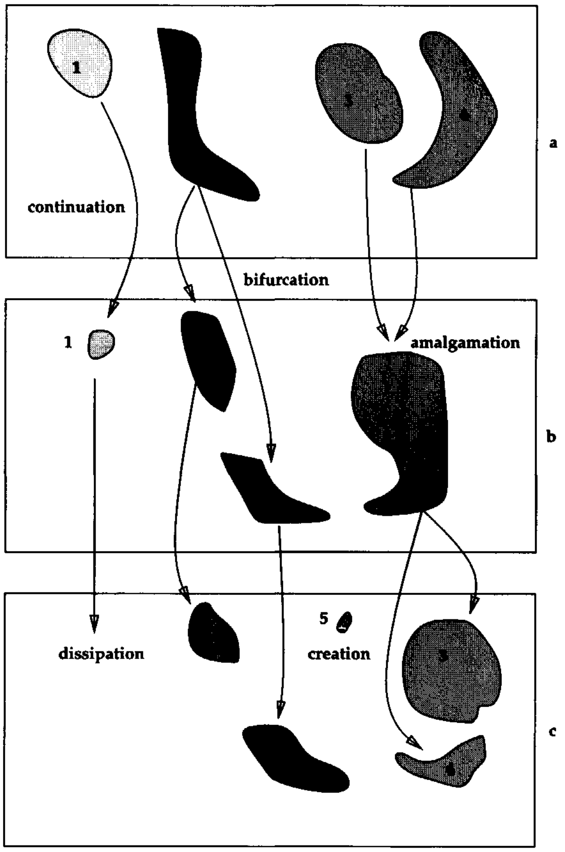


Figure 2.13: Tracking events: continuation, creation, dissipation, bifurcation, and amalgamation, [Samtaney *et al.*, 1994].

The criterion for correspondence between two features is that the difference between each attribute pair must fall within a certain tolerance. For continuation, the criterion for an attribute is:

$$attr(O_B^{i+1}) - attr(O_A^i) \leq T_{attr} \tag{2.7}$$

with O_B^{i+1} the object in frame $i+1$, O_A^i the object in frame i, and T_{attr} the tolerance value for a specific type of attribute. In case the attribute is a position, the distance between the two objects can be tested:

$$dist(O_B^{i+1}, O_A^i) \leq T_{dist} \tag{2.8}$$

In case the attribute is a scalar value such as a volume, the difference or the relative difference can be tested:

$$vol(O_B^{i+1}) - vol(O_A^i) \leq T_{vol} \tag{2.9}$$

$$\text{or:} \quad \frac{vol(O_B^{i+1}) - vol(O_A^i)}{\max\left(vol(O_B^{i+1}), vol(O_A^i)\right)} \leq T_{vol} \tag{2.10}$$

The user has to assign a certain tolerance value T_{attr} for each type of attribute that he wants to be tested.

Bifurcation is detected by testing the sum of attributes of the group of objects that result after bifurcation $\sum attr(S_{b\in N}^{i+1})$. Thus, the correspondence criterion for bifurcation is:

$$\sum attr(S_{b\in N}^{i+1}) - attr(O_A^i) \leq T_{attr} \tag{2.11}$$

with $S_{b\in N}^{i+1}$ the sum of the combination of objects in neighborhood N. The summation of attributes may be slightly different depending on the type of attribute: e.g. the position attribute is weighted by the volume or mass:

$$pos(S_{b\in N}^{i+1}) = \frac{\sum_b^N vol(O_b^{i+1}) \times pos(O_b^{i+1})}{\sum_b^N vol(O_b^{i+1})} \tag{2.12}$$

In this way the center of gravity of a combination of objects is calculated. The correspondence criterion for amalgamation is the inverse of the bifurcation criterion.

Also, [Samtaney *et al.*, 1994] introduced a schematic representation that shows the relations of features in successive frames by means of a directed acyclic graph (DAG). Each feature is represented by a node and the correspondences between features are shown by edges. Figure 2.14 shows a DAG history of features (the first number in each box is the feature number, and the second number is the frame number). Specific events are recognized by the number of edges that go in and out of a feature node.

Evaluation of the two methods

The method of [Sethi *et al.*, 1994] results in smooth trajectories corresponding to the preferences of the human visual system for continuity of motion. However, it uses the attributes as individual parameters in a k-dimensional property space where attributes are not related to each other and have no physical meaning. Furthermore, the greedy exchange method uses an optimization and therefore

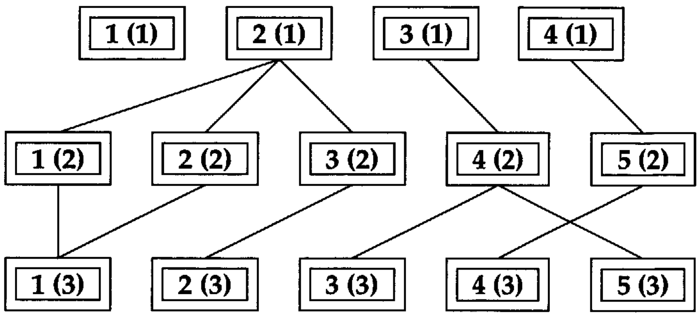


Figure 2.14: Directed acyclic graph (DAG) history of the evolution of features, [Samtaney *et al.*, 1994].

is computationally expensive, this is shown in Section 6.7.3. Also, the method cannot detect events.

The method of [Samtaney *et al.*, 1994] does treat the attributes as meaningful variables, and does detect particular events. However, the method lacks sense of continuity of motion and the events 'creation' and 'dissipation' are not explicitly detected; they are merely a result of failing to find a positive match.

In conclusion, the two methods present interesting ideas. Especially, [Samtaney *et al.*, 1994] is a precursor and a great source of inspiration to our work. The use of feature attributes for tracking presents opportunities for interactive tracking, and the concept of event detection brings new ideas for different types of events. The lack of continuity of motion could be compensated by means presented by [Sethi *et al.*, 1994], thus combining the advantages of both methods.

2.7 Thesis Preview

The research described in this thesis focuses on the following three main subjects:

- *Feature quantification and data representation.*

  A generic feature data representation is designed that allows the storage and manipulation of features with many different types of attributes (Chapter 3). Many of the features obtained by feature extraction techniques described above, can be represented by this feature data representation.

  The accuracy of the calculated feature attributes is tested for two attribute calculation methods (Chapter 4). Furthermore, a new attribute calculation method is presented that determines skeleton shape attributes, which allows the description of the geometric and topological structure of an object (Chapter 5).

- *Feature tracking and event detection.*

  The feature data representation allows the investigation of feature evolution in time-dependent data. Features are tracked based on their attributes only, i.e. the original grid data is not needed. The feature tracking method (Chapter 6) has a better sense of continuity of motion than the method of Samtaney, and treats attributes as related quantities instead of individual parameters in a k-dimensional property space (like Sethi). Also, the tracking method can track generic features; as long as they can be represented by our feature data representation.

  The feature tracking results in continuous paths of features that describe the evolution of objects. This evolution can be investigated for significant events by our event detection method (Chapter 7). We improved the event detection and classification described by Samtaney with a number of new events.

- *Visualization of evolving phenomena.*

  The results of the feature tracking and event detection are visualized by a linked combination of two views: the feature view and the graph view. The feature view shows the features in 3D space by iconic representations. The graph view shows the event graph which is the result of the feature tracking and event detection stage. With these two views the user can interactively investigate the tracking results and analyze the evolution of the features. The combination of these two views provide a more sophisticated visualization than the DAG-visualization by Samtaney.

The techniques described above are successfully applied to a number of case studies (Chapter 8).

Chapter 3

Feature Data Representation

The feature-based visualization techniques described in the previous chapter, result in a variety of features. For each different feature type and application domain, a different set of attributes may be calculated. Also, the attributes may differ depending on the feature extraction technique used, and depending on the interest and focus of the investigation.

The different foci of investigation results in a large variety of possible attribute types and collections of attributes. For example, the attributes describing a vortex feature may include its core and the dimensions of the vortex tube around the core, but also general attributes like its total volume, the center of gravity, the average rotational speed, and so on. All these attributes are optional; a selected set can be used to describe the vortex. Moreover, a different application consists of other types of features described by completely different sets of attributes.

Irrespective of the large variety of attributes, a need exists for generic techniques for the manipulation and the comparison of features. It should be possible to calculate with features without knowing their exact contents, i.e. without concerning ourselves with the underlying physical significance of attributes and features. The only restriction is that the features are of the same type and contain equal sets of attributes.

We introduce a generic *feature data* representation, which is a quantitative representation of the feature attributes and which provides a number of functions and operations for the processing of features. With the feature data representation, a feature becomes an entity that can be stored, manipulated, compared and visualized.

This feature data representation requires a carefully designed framework which is explained in this chapter. The structure of the chapter is as follows. First, we discuss the motivation and requirements for a feature data representation. Then, we give a description of the class hierarchy that is implemented for the representation of features. Finally, we describe the representation of features in time-dependent data, and the file format that is associated with it.

3.1 Framework for Feature Data

3.1.1 Motivation

A feature data representation must facilitate the further investigation, analysis and visualization of features. During the feature extraction step, the original grid data was analyzed and the focus was directed to the interesting features in these data. Now, we have data describing these interesting features and we want to investigate these data. For instance to track the features in time and investigate feature evolution. The correspondence problem can be solved using the feature data only; the original grid data are not needed for feature comparison.

The use of a feature data representation makes the original grid data dispensable because all relevant information about the interesting features in the grid data is (or at least should be) available in the feature data. For some purposes it may be helpful to revert to the original data, e.g. to find 'weak' features (see Section 6.8). For these purposes, it is essential that we can reproduce the features. The feature data representation should include information about the whole process, including parameters, of feature extraction and quantification.

The transformation from grid data to feature data results in a huge reduction of data size which facilitates the interactive analysis of features. Especially for time-dependent data this is a tremendous advantage because this type of grid data are usually very large. It becomes feasible to store a whole time series in main memory and go back and forth in time in several passes and still remain interactive. This exploration in time would be unmanageable with the original raw data. The ability to perform multiple passes is essential for the process of feature tracking and event detection, as will be discussed in Chapters 6 and 7.

Feature data allows the interactive visualization and exploration of features in time-dependent data. The icons associated with the features can easily be displayed in a 3D viewer. The user can interactively browse through the frames, look at the objects in 3D-space and see their evolution interactively. This has great advantages compared to the creation of 2D movies where the viewpoint is predefined and interactive navigation is not possible.

3.1.2 Requirements

A feature data representation has to cope with a dilemma between generalization and specialization. Feature data should provide a generic representation for all types of features, but the behavior of a specific feature should conform to the underlying physics of the application. Vortices in a flow will behave differently from nuclear particles in a magnetic field. We want a framework that can store both types of data, and still allow certain operations like tracking and event detection.

A solution for this dilemma is found in the 'overloading' of basic functions and operations on features. Certain basic functions and operations for the manipulation of features are defined for every type of feature, but the functions are overloaded for each specific type in a way that it acts in accordance with the underlying rules of the application. This will be explained in more detail in Section 3.2.

The overloading of functions and operations makes the framework flexible. It is easy to construct a new type of feature by creating a subclass, and overloading its member functions in accordance with the physical rules underlying the new type. The storage, manipulation, comparison, and visualization of the feature is acquired immediately. Processes like time tracking and event detection do not have to be adapted for every new type of feature because they use only these basic functions and operations. The new type of feature can immediately be tracked if the basic functions are defined.

3.1.3 Relevant functions and operations

The functions and operations associated with a feature should support a wide range of manipulations. It must be possible to calculate with features. For instance, to calculate a prediction in time by linear extrapolation:

$$F_{i+1} = F_i + \frac{(F_i - F_{i-1})}{(t_i - t_{i-1})} * (t_{i+1} - t_i) \tag{3.1}$$

where F_i is the feature at time step i, and t_i is the time at step i. In order to perform this extrapolation, we must be able to add and subtract features, and be able to multiply a feature by a scalar number.

Table 3.1 lists a number of relevant functions and operations on features. They include general functions for input/output (`Read` and `Write`), copying (`Copy`), numerical calculation (`Subtract`, `Add`, `Times`, and `Divide`), and visualization (`Icon`). Other functions are especially created for the feature tracking and event detection procedures. The `Merge` function merges a number of features into one, `CorrespondenceFunctions` returns a list of correspondence functions that can be used to compare two features (Section 6.2.1), a number of functions exist that return a list of event functions for the detection of a particular event (Chapter 7). In addition to these functions, many other convenient functions were created for the access of the data within a feature.

This list of functions is adequate for our purpose of feature tracking and event detection, but new functions and operations may be added when desired.

Function name	**Description**
`Read, Write`	Read or write feature data from or to file.
`Copy`	Create a copy of this feature.
`Subtract, Add`	Subtract or add the attributes of two features. This equals the $+$ and $-$ operators.
`Times, Divide`	Multiply or divide the feature attributes by a scalar value. This equals the $*$ and $/$ operators.
`Icon`	Return an iconic geometry for visualization.
`CorrespondenceFunctions`	Returns a list of correspondence functions for the comparison of two features.
`ExitFunctions`	Returns a list of exit functions for the detection of an exit event.
`DeathFunctions`	Returns a list of death functions for the detection of a death event.
`TopoFunctions`	Returns a list of topological functions for the detection of a topological event.
`NeighborhoodFunctions`	Returns a list of neighborhood functions to establish the neighborhood between two features.
`Merge`	Creates a 'merged' feature from a number of features that merge into one.

Table 3.1: Relevant functions and operations that must be defined for a feature, they include general functions for I/O and copying, numerical operations, a function for visualization purpose, and specific operations for feature tracking and event detection.

3.2 Class Hierarchy

The feature data representation is implemented as a C^{++} class-library. The data structure exploits the concepts of inheritance and overloading of functions from parent classes to children classes. A child inherits all the characteristics from its parent, but may overload certain functions to match its own characteristics. For instance, the parent class `Attribute` has a virtual function `Read` that reads the data from file. Each child derived from `Attribute` overloads this function according to its own file-format. Now, we can call the function `Read` for any `Attribute` without knowing the specific type of the child.

The structure of the class hierarchy is designed following the feature extraction pipeline shown in Figure 2.9.

The attributes are calculated by a calculation method, and it is possible to use more than one calculation method to obtain attributes describing different aspects of the feature. Each calculation method results in a group of attributes that are often closely related to each other. The attributes by themselves provide information about the feature, but together they provide more information. Therefore, the group of attributes resulting from one calculation method are collected in an *attribute set.*

As an attribute set, the attributes have a stronger significance than each attribute separately. For instance, an ellipsoid fit consists of three attributes: position, axis lengths, and axis orientations. The orientation attribute only has a significant meaning in combination with the axis lengths and the position.

The building blocks of the attribute sets are the attributes. They contain the basic values describing the characteristics of a feature. There are different data types for an attribute, e.g. scalar, vector, matrix, string, and index. However, each type is subdivided into a number of specific types of attributes which gives a specific meaning to the values. For example, a position attribute is a vector of three coordinates in 3D space, as a position attribute the three values are coordinates in space, but as a vector they have no significant meaning.

With the basic classes of calculation method, attribute set, and attribute, we can construct the feature data representation describing a feature. The constructor of a feature has as input the list of calculation methods that were used for quantification. Each calculation method results in an attribute set which holds a set of specific attributes. We designed a data structure that specifies the subclasses for each of these basic classes.

3.2.1 Attribute

Figure 3.1 shows the hierarchy of the `Attribute` classes. The parent is the `Attribute` class which provides a number of basic, abstract functions that may be overloaded by the children. Overloading changes the implementation of such a function so that it depends on the semantics of the derived class.

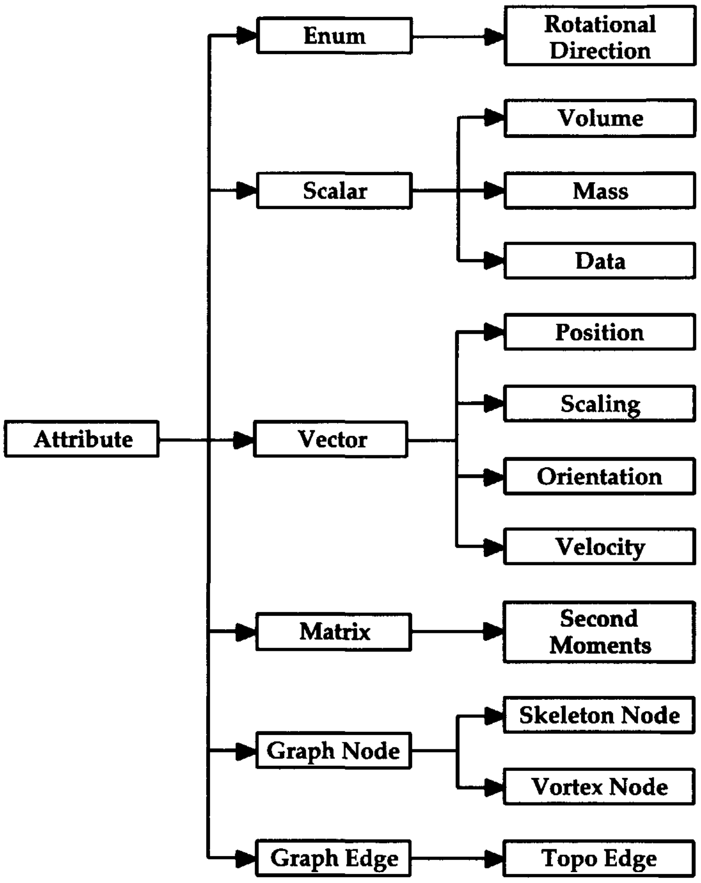


Figure 3.1: The hierarchy of the `Attribute` classes.

The second level in the hierarchy is a number of attribute data types such as matrix, vector, and scalar attributes. They do not yet have a specific physical meaning, but provide a general implementation for the storage of the data and for the implementation of certain member functions. For instance, the class `Vector` contains a vector of floats. The member function that reads a vector from a file is implemented for `Vector` and does not have to be re-implemented for its children. The same principle applies for other operations and functions that may be executed on vectors. In this way, much functionality is passed to children by inheritance.

At the third level the attributes get a specific physical meaning. `Position` contains the three coordinates in space, `Volume` the volume of the object, and `Skeleton Node` a skeleton graph node (see Section 5.4). In this way the values in the attribute obtain a higher semantic level, instead of being just a set of values.

The hierarchy of attribute classes provides a general implementation for all kind of attributes. New types of attributes can be easily added by adding a new class and overloading the specific member functions. The overloading should agree with the rules that apply for that specific type of attribute.

3.2.2 Attribute set

An attribute set is a set of related attributes that result from a specific calculation method. For instance, the `Ellipsoid` attribute set is related to the ellipsoid fitting calculation method. It consists of three attributes: `Position`, `Scaling`, and `Orientation`.

Often it is possible to calculate derived attributes from an attribute set. An example is the volume calculated from the axis lengths in the ellipsoid data:

$$V = \frac{4}{3}\pi r_1 r_2 r_3 \tag{3.2}$$

This shows that an attribute set may hold more information than what is directly available. Sometimes it is even possible to derive a complete attribute set from another attribute set. For instance, the `Ellipsoid` attribute set can be derived from the variance/covariance matrix stored in the `Spatial Distribution` attribute set (see Section 4.1.2).

Figure 3.2 shows the structure of the `Attribute Set` class hierarchy. Similar to the `Attribute` hierarchy, three levels can be recognized: `Attribute Set` is the basic, abstract parent class, then some generic children are derived from the attribute set, and finally a set of specific attribute sets are derived.

For each specific attribute set the functions and operations to calculate with attributes are defined. These functions include the basic arithmetic operations: addition, subtraction, multiplication, and division of attribute sets. The rules that apply for each of these functions depend on the type of the attribute set. These functions allow the calculation with feature attributes.

Sometimes two types of attribute sets hold the same set of attributes, but have a different meaning. The meaning of the attributes depend on the calculation method that is the foundation for the attribute set. For instance, the attributes in the ellipsoid (`Position`, `Scaling`, and `Orientation`) are similar to the oriented bounding box. Although the attribute types are the same, the set as a whole has a different meaning and may act differently with certain operations. It is this functionality that is implemented in the specific subclasses of `Attribute Set`.

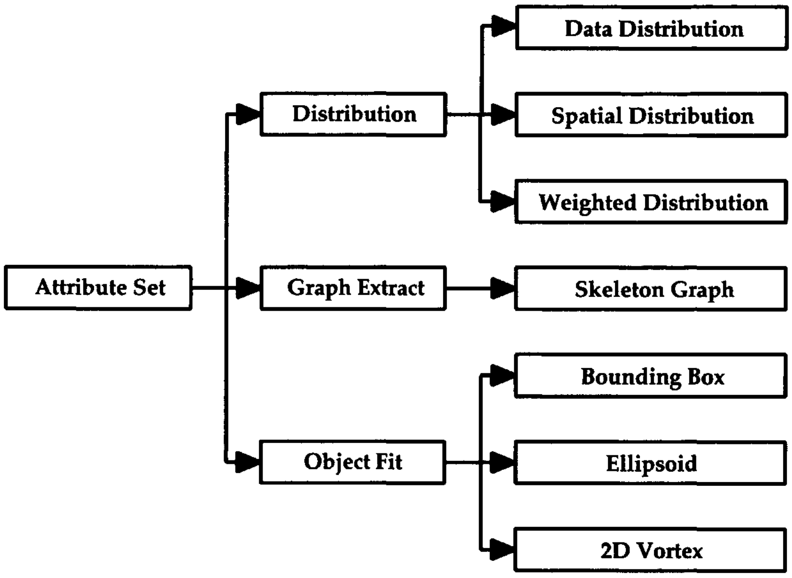


Figure 3.2: The hierarchy of the `Attribute Set` classes.

Each attribute set has a member function `Icon` that returns an iconic geometry for visualization purposes. The correct mapping of the attributes on a parametric iconic object ('Iconic Mapping' in the feature extraction pipeline, Figure 2.9) is achieved by overloading this function. Often a choice can be made between several possible icon types.

The geometric objects that are returned by `Icon` are defined in Open Inventor [Wernecke, 1993]. Open Inventor is a well known 3D Computer Graphics library based on OpenGL [Woo M. and Neider J. and Davis T. and Shreiner D., 1999] and that can be accelerated by graphics hardware. In this way, a fast display can be expected on a workstation with hardware support for OpenGL output.

3.2.3 Calculation method

A calculation method calculates attributes by applying a model to a feature extracted from the raw data. The result is one specific attribute set that should act according to the rules underlying the model of the method. The relationship between attribute set and calculation method is a one-to-one relation. This can also be seen when we compare Figure 3.3 and Figure 3.2. A `Volume Integral` results in a `Distribution`, an `Object Fitting` results in an `Object Fit`, and so on. The `Calculation Method` class is the 'method' equivalent of the 'data' class `Attribute Set`.

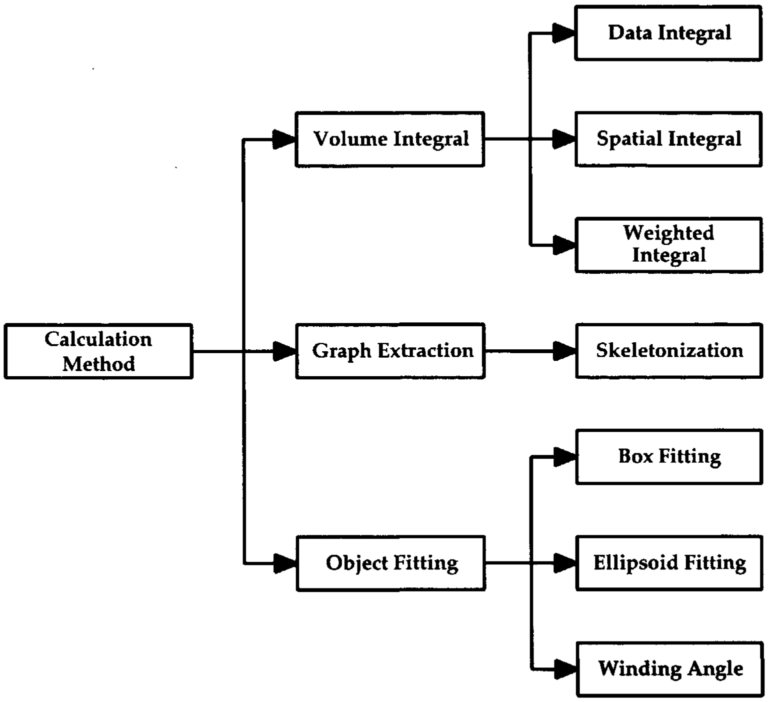


Figure 3.3: The hierarchy of the `Calculation Method` classes, which is isomorph to the `Attribute Set` hierarchy of Figure 3.2.

The model underlying the calculation method includes the evolutionary characteristics of a feature. Some calculation methods produce information that allow certain types of events. For instance, `Skeletonization` calculates topological information of a feature, so that topological events may be detected. But this information is not available when `Ellipsoid Fitting` is used, so that this event cannot be detected. The behavior of a feature is determined by the calculation methods used in the feature extraction step.

Each calculation method is associated with a number of event functions for each specific type of event (see Chapter 7). If no event function is associated to this method for a specific event, then that event cannot be detected as far as this method is concerned. But the event can still be detected if another method does provide an event function for that type of event.

Because the calculation method and attribute set are related one-to-one, this also establishes the behavior of the corresponding attribute set. The corresponding attribute set should be implemented in accordance to the specific rules underlying the model of the calculation method.

3.2.4 Feature

The class Feature embodies the feature data representation describing the feature characteristics. The class hierarchies described above, are used in the following way.

Formally, a feature should include all information that is needed in order to reproduce it from the original grid data. This includes a reference to the raw data, and every parameter of the complete process of feature extraction. However, it is impossible to represent every feature extraction technique since so many different techniques exist. Here, a Feature is constructed by a list of calculation methods as input. The methods determine what models are applied to the feature, and thus they determine the type and the behavior of the feature, including the detectable events in temporal evolution.

Each calculation method is associated with an attribute set that is also stored in the Feature. The arithmetic operations performed on the features are actually performed on these attribute sets because they hold the feature attributes. Furthermore, the feature can be visualized by the icons that are associated to each attribute set, where the attributes in the attribute set are mapped onto the parameters of the icon. The user can choose one of the icons in order to investigate certain attributes.

Next to the calculation methods and the corresponding attribute sets, the Feature has a FeatureNr and an ObjectId. The FeatureNr refers to the index number in a frame, i.e. the number obtained during clustering. The ObjectId refers to the object identification of the tracked object in time, i.e. ObjectId $= -1$ until the feature is tracked and added to an object path as described in Section 6.3.

With this class structure, a feature becomes an entity that can be stored, manipulated, compared and visualized.

3.3 Time-Dependent Feature Data

Features can be extracted from each frame of the time-dependent data, and the attributes may be stored as feature data. In order to store the complete time series of feature data, the classes Frame and TimeSet are created.

The class Frame holds the collection of features in a single time step. It holds a list of Feature objects, a frame number for identification, and the time stamp. With this class we can store a complete collection of features at a specific time, and we can manipulate them easily.

The collection of all frames is stored in a class called TimeSet. It holds a list of frames and information about the original grid data, such as the dimensions, and a system boundary description. TimeSet can keep the complete time-dependent feature data in main memory, which allows random order access operations and multiple pass searching over the frames.

A simple file format has been designed for the time-dependent feature data representation. Table 3.2 shows a feature data file with ellipsoid data for each feature. The data consists of 100 frames, the original grid data was defined on a grid with dimensions 128 × 128 × 128, and the system boundary is a single rectangular box of 128 grid cells in each direction. The real feature attributes are written after this heading. Frame 1 consists of 28 features, feature 1 has one attribute set, the ellipsoid, and values are given. Then the information is given for feature 2, and so on, until every feature in every frame is defined.

```
NrOfFrames: 100
Dimensions: 128 128 128
SystemBounds: 1
  Box 0 128
      0 128
      0 128

Frame 1: 28
  Feature 1: 1 Ellipsoid
    (x, y, z) (r1, r2, r3) (a1, a2, a3)
  Feature 2: 1 Ellipsoid
    (x, y, z) (r1, r2, r3) (a1, a2, a3)
  .
  .
  .

Frame 2: 31
  Feature 1: 1 Ellipsoid
    ...
  .
  .
  .
```

Table 3.2: The file format of the time-dependent feature data representation.

Chapter 4

Accuracy of Attribute Calculation

Feature data can only be used for further analysis if the accuracy and stability of the calculated attributes has been verified. Attributes are only useful if a certain precision can be guaranteed. We need to make sure that the attributes give an accurate description of the characteristics of a feature.

The accuracy and stability of the attributes depend on the method that is used during the attribute calculation step in the feature extraction pipeline, see Figure 2.9. The attributes are calculated from a cluster of selected grid points indicating the region of interest. In general, the accuracy and stability is larger when the calculation method uses more points in the cluster. For instance, an average data value is more stable than the maximum of a data value.

The question is: how accurate is the method, and how sensitive is it to small variations. The problem definition is described in more detail in Section 4.2.

In this chapter, we apply experimental methods to demonstrate the accuracy of the feature data. The chapter discusses an experimental sensitivity analysis for the verification of the accuracy and stability of attribute calculation methods. With this analysis, two calculation methods are tested with respect to different noise levels, to different grid densities, and to different feature extraction parameters. In this way, a number of guidelines are derived for working with the feature extraction method in practice. The work described in this chapter was also published in [Reinders *et al.*, 1998].

4.1 Attribute Calculation Methods

Attribute calculation determines the attributes to characterize the features. There are many methods to calculate attributes that also depend on the feature extraction techniques used. Here, two methods are discussed that are particularly useful for 3D amorphous features with a certain position and size. The first is the calculation of volume integrals, and the second is the determination of the best ellipsoidal fit.

4.1.1 Volume integrals

The volume integral is a generic method that can be used to calculate many different aggregate attributes such as center position, volume, mass, and average data values. Basically, the volume integral can be approximated by a weighted sum over the selected points of a cluster. Each point contributes to the result of the volume integral. Since the volume integral uses all points in the cluster, we expect that the calculated attributes are accurate and stable.

Volume integrals allow the calculation of global attributes that describe aggregate characteristics of a feature. Examples are the average values and the variance/covariance matrices of position, data values, and weighted values. If S is a cluster of selected points, $\mathbf{x}$ a position of a point in the cluster, ρ the density value at $\mathbf{x}$, $\mathbf{v}$ a vector value at $\mathbf{x}$, and $\mathbf{v}^T$ the transpose of $\mathbf{v}$, then the following quantities can be calculated with volume integrals[1]:

Volume of S (V_S)	$\int_S dS$
Center position of S	$\frac{1}{V_S}\int_S \mathbf{x}dS$
Second moments of S	$\frac{1}{V_S}\int_S \mathbf{x}\mathbf{x}^T dS - \frac{1}{V_S}\int_S \mathbf{x}dS \frac{1}{V_S}\int_S \mathbf{x}^T dS$
Mass of S	$\int_S \rho dS$
Center of gravity of S	$\frac{1}{V_S}\int_S \rho\mathbf{x}dS$
Average $\mathbf{v}$ of S	$\frac{1}{V_S}\int_S \mathbf{v}dS$
Var/cov matrix for $\mathbf{v}$ of S	$\frac{1}{V_S}\int_S \mathbf{v}\mathbf{v}^T dS - \frac{1}{V_S}\int_S \mathbf{v}dS \frac{1}{V_S}\int_S \mathbf{v}^T dS$

Section 5.2 of [van Walsum, 1995] shows how these integrals can be approximated on curvilinear grids using quadrature rules (see also Section 4.5 of [Press *et al.*, 1992]). The integral $\int f(x)dx$ is approximated by evaluating $f(x)$ at sample points (abscissae) $x_1 \ldots x_n$, so that

$$\int_S f(x)dx = \sum_{i=1}^{n} W_i f(x_i) \tag{4.1}$$

with W_i the weight factor for position x_i. The type of quadrature determines the number of samples, the sample positions, and the values for the weights W_i. Each sample has a contribution to the volume integral that depends on the weight, and the interpolated data value at that point. The interpolated data

[1] N.B. The second moments of S is equal to the variance/covariance matrix of the point positions, i.e. the point distribution in space.

value is calculated by tri-linear interpolation. Figure 4.1 shows the four-point Gauss-Legendre quadrature around a selected point. Other quadrature rules are the trapezium rule, and the two-point Gauss-Legendre rule.

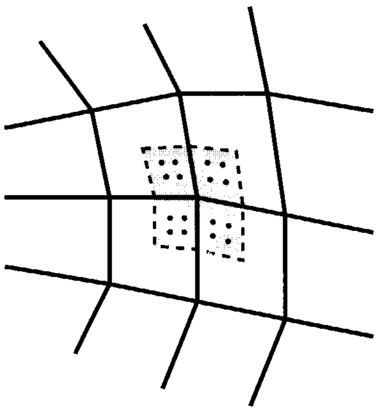


Figure 4.1: The four-point Gauss-Legendre quadrature for the integration based on selected points.

Integrals using trapezium rules do not need interpolation of the data, only the point values are used. Linear functions, like averages, in x, y and z can be integrated exactly with the trapezium rule. To integrate higher order functions (such as second moments, variance, and covariance), higher order quadratures are needed. The higher order quadratures require more computation time, but are more accurate. The accuracy of the derived attributes depend on the type of quadrature rule used in the integration method.

4.1.2 Ellipsoid fitting

Like the volume integrals, object fitting is a generic technique for the calculation of attributes. An object with a limited number of degrees of freedom is fitted as accurately as possible around the geometry of a cluster of selected points. Examples of fitted objects are the bounding box, the oriented bounding box, and the ellipsoid.

Ellipsoid fitting is a well-known method that provides a first order approximation for the geometric shape of an object [Fung, 1965; Haber & McNabb, 1990; Silver *et al.*, 1991]. The number of degrees of freedom is nine: three for the center point position, three for the axis lengths, and three for the orientations of the ellipsoid axes. In [Silver *et al.*, 1991] the ellipsoid attributes were used to describe the temporal evolution of features in time.

The ellipsoid fit provides a good indication for the global geometry of an object; position, size, and orientation. The center point position is the average position of the cluster points. The volume or size of an object can be determined from the axis lengths (equation 3.2). If the ellipsoid has a high eccentricity, i.e. one of the axis lengths clearly differentiates from the others, the direction of this axis provides an indication of orientation.

Ellipsoid attributes can be calculated from the second moments, or the variance/covariance matrix of a vector data value, see the list of volume integrals[2]. The ellipsoid fitting derived from the second moments results in attributes describing the geometric distribution of the cluster, while the ellipsoidal fitting derived from the variance/covariance matrix of a vector data value describes the distribution of that vector data value.

The ellipsoid attributes are obtained by solving the eigenvalue problem for the second moments or variance/covariance matrix A:

$$A\mathbf{x} = \lambda\mathbf{x} \tag{4.2}$$

where $\mathbf{x}$ is an eigenvector, and λ is the corresponding eigenvalue. If the matrix A is not degenerate or defective, the solution to this problem will result in three eigenvalues and eigenvectors. The eigenvectors correspond to the directions of the three axes of the ellipsoid, while the eigenvalues are proportional to their lengths. The eigenvalue problem is solved using the Cyclic Jacobi algorithm, an iterative diagonalization process, see Section 11.1 of [Press *et al.*, 1992].

4.2 Problem Definition

When calculating the attributes, accuracy issues arise. Here, we make an inventory of possible error sources, and their influence on the accuracy and stability of the attribute calculation.

Errors can be introduced in two ways: errors in the calculation procedure, and errors in the data values. The calculation procedure introduces errors because it has to deal with discrete data. The precision of the attributes depend on the numerical method used for calculation. Errors in the data values result in small variations in the cluster of selected points. A good attribute calculation method is insensitive to these variations and results in stable attributes.

4.2.1 Accuracy of the procedure

The discrete data is a discontinuous approximation of the continuous space; the 3D space is represented by a finite number of discrete sample points. Numerical methods, like attribute calculation, have to deal with this discrete data and

[2]N.B. The calculation of an ellipsoid fit includes the calculation of volume integrals (the average value and the variance/covariance matrix).

reconstruct the real data values as accurate as possible. For that purpose they use procedures like sampling, convergence, and interpolation. The numerical methods can use linear, or higher order procedures for the calculation of data values. Higher order procedures are more accurate, but are computationally more expensive. There is a tradeoff between precision and computational costs.

The accuracy of the attributes depend on the precision with which the calculation method can calculate the attributes from the discrete data. The precision depends on the procedure that is used. For instance, the integration procedure using Trapezium or Gauss-Legendre quadratures, or the diagonalization procedure using the Cyclic Jacobi algorithm. It is important to know the precision of the procedure and to know when the precision compensates the computational costs of a higher order procedure.

The precision of a calculation method is also a function of the resolution of the grid. A dense grid means that the feature consists of a large number of grid points, which results in more precise attributes. The precision is better because the calculated attributes are determined from more elementary values. For example, the average position can be calculated more accurately when a larger number of grid positions is concerned. It is essential to know the minimal number of grid points that is required to calculate certain attributes.

4.2.2 Stability of the procedure

The stability of the calculation method is determined by the sensitivity of the method to small variations in the data values. The variations in the data values may be caused by noise in the data, missing data, or errors in the data acquisition (the simulation process, or the measurement device). These variations may change the data at a certain grid point in such a way that the outcome of the selection criteria is changed. This will especially be true for points where the data is close to the selection thresholds ('weak' data points). Grid points may be added or removed from the cluster of selected points, thus changing the attributes of a feature.

Here, we investigate the stability of the procedure with regard to noise. The presence of noise in data introduces false positives, and false negatives in the selected points. The effects on the calculated attributes depend on the number of points in the cluster. Volume integrals are more stable when they are based on a larger number of points. The stability is better because the average position of a large cluster changes little when an additional point is included, while the average position of a small cluster may change considerably when an additional point is included. The number of points is directly related to the size of the object and the (local) resolution of the grid.

Besides an error in the attributes, noise may cause the emergence of spurious features. Small clusters of selected points may appear when noise is added.

The number of significant features should be stable with respect to noise. The spurious features can be eliminated by choosing the right extraction parameters.

The extraction parameters

The extraction parameters consist of the selection threshold value, the cluster threshold, and the connected component definition. These can be chosen in such a way that the effects of noise are reduced.

- The *selection threshold* T_s value (or multiple values) decides whether the data in a grid point satisfies our selection criterion. It can be set above the noise level in order to eliminate noise effects, but this will also influence the resulting feature.
- The *cluster threshold* T_c is the minimum number of points in a cluster; all clusters with less than T_c number of points are discarded. Only large substantial features remain, and small features resulting from noise are removed. However, we may also remove small but genuine features that may be significant, e.g. with data values well above the selection threshold.
- The *connected component* definition can be defined as (provided that we have data on a structured grid): face-connected (where a point has 6 neighbors), edge-connected (18 neighbors), and vertex-connected (26 neighbors). This definition is crucial in the clustering stage, since it determines if two adjacent points are in the same cluster or not. Obviously, face-connected will result in more and smaller clusters than edge- or vertex-connected. The connected component in combination with the cluster threshold can be used as a tool to eliminate noise effects.

The extraction parameters must be chosen with care, especially in case of noisy data. Therefore, we will try to establish a number of guidelines for finding the right settings for the extraction parameters.

Recapulating, the accuracy and stability of the attributes depend on the noise in the data, on the number of points in a feature, and on the arithmetic method used for attribute calculation. It is vital to verify the reliability of attribute calculation with respect to these issues.

4.3 Experimental Setup

We use a simulation study to experimentally verify the accuracy of attribute calculation. The simulation study is designed as follows. Synthetic data is generated with synthetic features, i.e. with known attributes, and with noise that has a known distribution function. This data is used as input in the feature extraction pipeline. The obtained attributes are compared to the initial settings of the attributes. The difference between the initial and obtained attributes provides a measure for the accuracy of the feature extraction method.

The simulation study is performed with different noise levels, with different grid densities, and with different feature extraction parameters.

4.3.1 Synthetic data

Synthetic data is well-defined data that contains features with known attributes. The data is defined on a regular grid with a variable density. On the grid points a scalar field is created that can be controlled as follows. An initial value is given to every point in the grid, by default the initial value is set to zero, $V_i = 0.0$. On top of this initial value it is possible to add noise with a certain distribution function. We used a Gaussian distribution function with zero mean, and a variable standard deviation (SD); which is generated with an algorithm given by [Press *et al.*, 1992].

Features with known attributes are added on top of these initial data values. The user can specify an ellipsoid with given attributes: center position, axis lengths, and axis orientations and a data value at the center position of the ellipsoid ($V_c = 100.0$ by default). Data values are added to every grid point inside this predefined ellipsoid. The additional value depends on the position of the grid point inside the ellipsoid. The value decreases linearly from over a line through the center position to the surface, where the additional value is zero:

$$V_p = V_c \times \frac{\mathrm{d}(p,s)}{\mathrm{d}(c,s)} \tag{4.3}$$

with V_p the data value added at point p on the line, V_c the data value defined at the center, $\mathrm{d}(p,s)$ the distance between point p and the surface, and $\mathrm{d}(c,s)$ the distance between the center and the surface.

Figure 4.2 shows the histogram of the data within an ellipsoid feature. The background with data $V_i = 0$ is omitted from the histogram and no noise was added. Most of the points in the feature have a value close to zero and only few come close to the maximum value of $V_c = 100$. The situation is idealized: we have one feature with one maximal data value at the center. The feature cannot be mistaken for one of the 'noisy features', and it cannot break up into pieces.

The addition of noise will most strongly affect the feature-points near the surface of the ellipsoid, because here the data values are small. Still it is possible to extract the feature as long as the maximum data value of the feature is significantly higher than the noise data. This is shown in Figure 4.3, where a selection is made of points with a data value $> 2 * SD = 30.0 = T_s$. The figure shows the selected points by small cross-marks, and ellipsoids fitted around each cluster with two or more points. One of the ellipsoids is significantly larger than

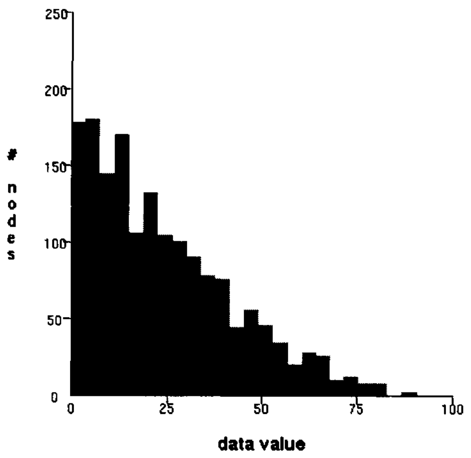


Figure 4.2: Histograms of the generated synthetic data (without noise) within one ellipsoidal feature.

the rest, this is the synthetic feature, it can be filtered out by choosing a larger cluster threshold, thus eliminating all small clusters.

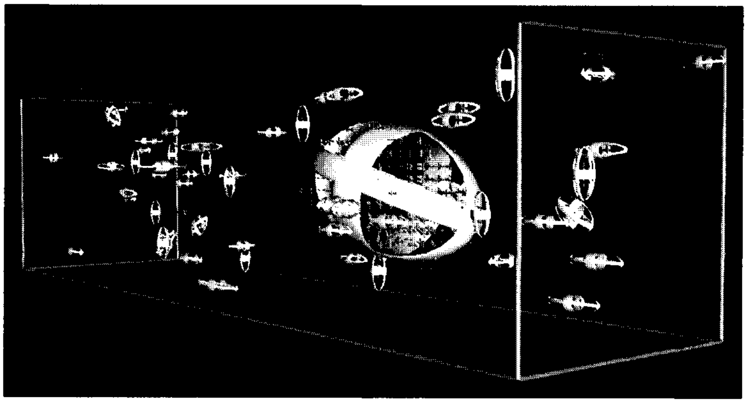


Figure 4.3: Features extracted from synthetic data with noise.

4.3.2 Experiments with the ellipsoid fitting method

This section describes the experiments that have been performed with the ellipsoid fitting method. These tests verify the accuracy of the volume integrals as well, because the ellipsoid calculation method includes the calculation of the average position and the second moments. Similar experiments can be executed on other attribute calculation methods.

Accuracy experiments

The accuracy of the method is tested by generating synthetic data without noise. In case of perfect accuracy, the obtained attributes should be exactly equal to the predefined ones. Errors depend only on the calculation method itself and on the density of the grid.

Every attribute can be tested separately, i.e. for the ellipsoid fitting method: center position, axis length, and axis orientation.

- **Center position.** The error in the detected position is expected to be smaller than a cell-size, this can be verified by the following experiment. Synthetic data is generated with a spherical feature with a fixed radius and a position moving in 50 steps from a corner point of a cell to the center of the cell. Each of the 50 data sets are analyzed by the feature extractor, and the resulting positions are used for error estimation.

We expect the resulting position to move stepwise through the cell, the steps are caused by points entering or leaving the moving sphere. This effect is illustrated in Figure 4.4. The distance to the diagonal (the real position) divided by the diagonal length, gives an error measure for the position, relative to the grid-cell size. The same experiment can be repeated for different grid resolutions, i.e. a feature with a larger number of selected points.

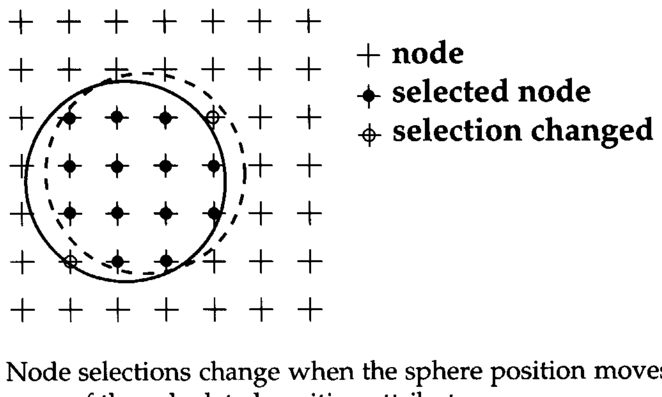


Figure 4.4: Node selections change when the sphere position moves, causing a stepwise change of the calculated position attribute.

- **Axis length.** To determine the accuracy of the axis lengths, synthetic data is generated containing ellipsoids with fixed orientation and position, and with the length of one of the axes varying in one direction. Again, the variation is limited within a cell, and the experiment is repeated for several grid resolutions. Errors are calculated relatively to the cell size.

- **Axis orientation.** Synthetic data is generated with ellipsoids with fixed position and axes ratios with an eccentricity of $3 : 1 : 1$, and with varying orientation of the main axis (from 0 to 45 degrees), for several grid resolutions. Errors are calculated relative to the maximum angle, i.e. 45 degrees.

The results of the experiments described above can be found in Section 4.4.1.

Stability experiments

As discussed in Section 4.2, there are three important parameters in the extraction procedure: the selection threshold, the cluster threshold and the connected component definition. The following experiments establish the relationships between these parameters, and the effects on the extracted attributes.

- **The selection threshold value.** Noise may introduce spurious clusters if the selection threshold value is set too low. The following experiment counts the number of clusters found, as a function of the threshold value, and of the noise level.

Synthetic data is generated with one feature, and for a number of different noise levels. Using this data, the number of clusters is monitored while slowly increasing the selection threshold until only one cluster (the synthetic feature) is found. The lowest threshold value that results in one feature is called the cut-off threshold value. It is an important value since it gives us the minimum threshold value that distinguishes the feature from the noise. The cut-off should be as low as possible, as higher threshold values result in smaller features. Therefore, the cut-off threshold will be used in further experiments, because it depends not only on the noise level, but also on the other extraction parameters.

- **The cluster threshold.** The cluster threshold is a useful parameter, since small clusters are removed by it. In many cases (especially if noise is involved) the selection results in single unconnected points that just happen to satisfy the selection criteria, but are not significant. The cluster threshold is often an adequate remedy to filter out these undesirable features. Therefore, we determine the cut-off threshold for different noise levels, as a function of the cluster threshold.

- **Connected component definition.** The connected component definition will affect the number of clusters found. The face-connected is more strict than the others, and will result in more and smaller clusters, which amplifies the effects of the cluster threshold. In order to test this, the cut-off threshold is determined for all three definitions as a function of the cluster threshold, for one given noise level.

4.4 Results for Ellipsoid-Fitting

4.4.1 The accuracy of the method

First, the accuracy of the ellipsoid fitting method is established. Figure 4.5 shows the results of the accuracy tests for the position detection. The center position of the sphere starts at one corner point of the cell (relative position = 0), and ends in the center of the cell (relative position = 0.5). The obtained position is plotted as a function of the input position. It changes discontinuously every time a point enters or exits the moving sphere, hence a stepwise update of the position is found. A preliminary experiment showed that results are symmetrical from 0.5 to 1.0.

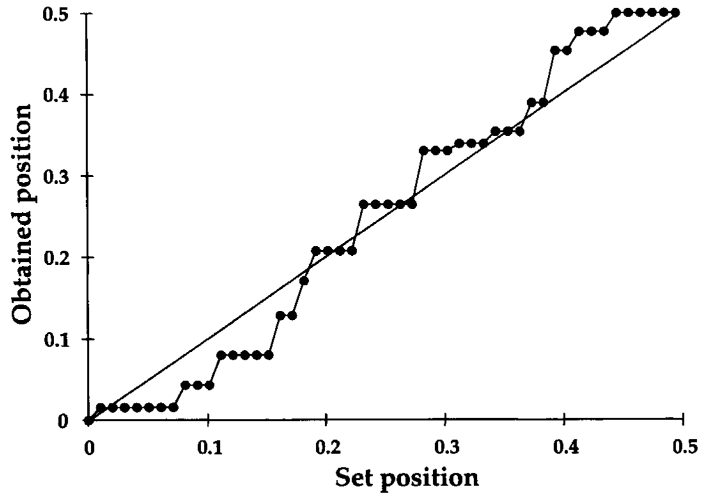


Figure 4.5: Stepwise movement of the position within a cell.

The average distance to the diagonal is considered the error in the position detection. The error is relative to the grid cell size, an error of 10% means an error of 10% of the grid cell size. Similar stepwise results are obtained for the axis lengths and orientation detection. Since the attributes were varied within cell size, we may conclude that the ellipsoid-fit method detects shifts with a sub-cell accuracy.

As may be expected, the accuracy improves when the grid density increases. Figure 4.6 shows the errors of the ellipsoid attributes as a function of the number of selected points in the cluster. The figure clearly shows the exponential decrease of the error with the number of points. The errors are below 7% (of the grid cell size) when the clusters consist of more than 15 points. We believe this is acceptable.

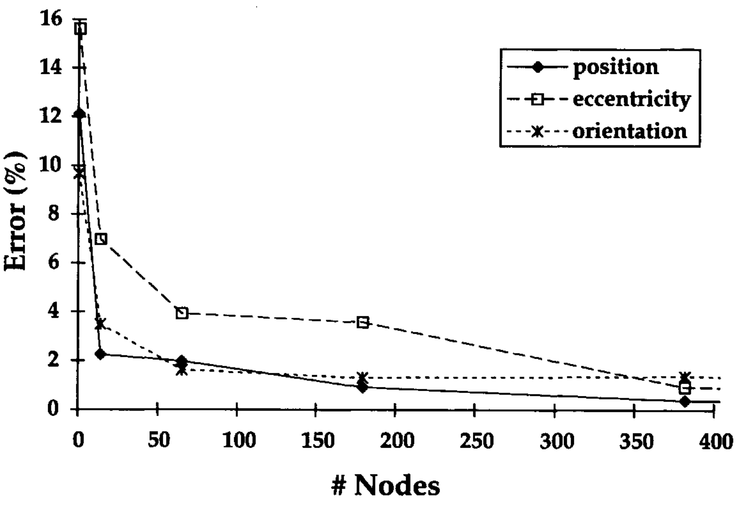


Figure 4.6: Errors for the ellipsoid attributes obtained by experimental accuracy tests.

4.4.2 Stability of the method

Now that the accuracy has been assured, the stability of the method with noisy data is investigated.

First, the number of clusters is determined as a function of the selection threshold value, and as a function of the noise level. Figure 4.7 shows that for low thresholds a large number of clusters is found. This is an obvious result since the noise causes many point values to rise above the threshold level. If noise is added with an $SD > 10$, the number of clusters first increases with increasing threshold values because many points connect to form a large cluster which breaks up as the threshold is increased. In the end, a threshold value is found where only one feature remains. This value is the *cut-off threshold*: it is the selection threshold value where the single 'genuine' feature remains and all spurious small features due to noise disappear. The cut-off threshold becomes higher as the noise level increases.

The cut-off threshold is examined as a function of the noise level and the cluster threshold. Figure 4.8 shows that the cut-off threshold increases with increasing noise, yet the cut-off threshold remains low for a large cluster threshold. Using a cluster threshold of at least 20 points, it suffices to use a selection threshold value of $1 * SD$ in order to eliminate all clusters due to noise. If smaller features are expected, then a cluster threshold of 5 points in combination with a selection threshold value of at least $2 * SD$ suffices, provided that the threshold value is significantly smaller than the maximum data value in the feature.

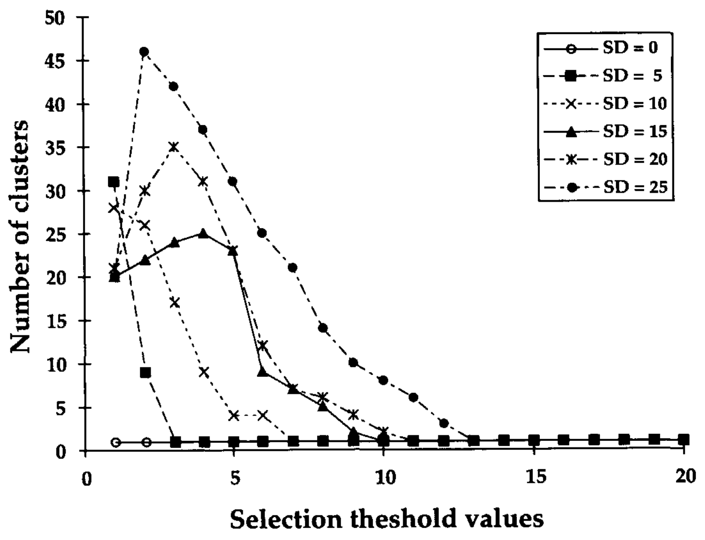


Figure 4.7: The number of clusters as a function of the selection threshold value and the noise level.

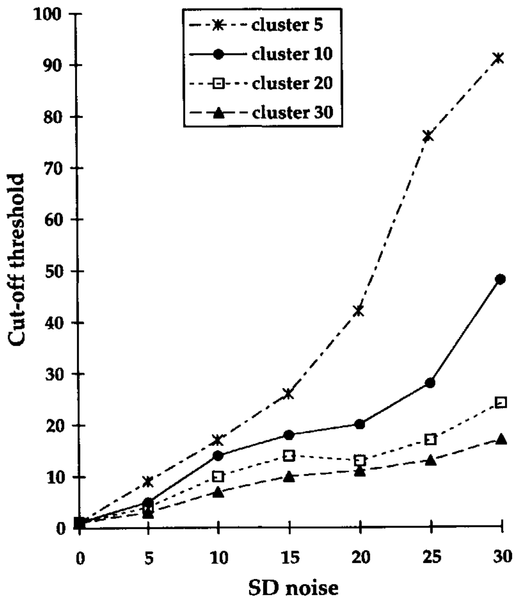


Figure 4.8: The cut-off threshold value as a function of the noise level and the cluster threshold.

The next experiment determines the cut-off threshold for the three connected component definitions, as a function of the cluster threshold, with the noise level set to $SD = 15.0$. The result is plotted in Figure 4.9, which shows that the cut-off threshold drops as the cluster threshold increases, and also that it drops more sharply if the connected component definition is set to face-connected. This definition results in more, smaller clusters which are easier to eliminate by the cluster threshold. Therefore, we prefer to use the face-connected component definition in case of noisy data.

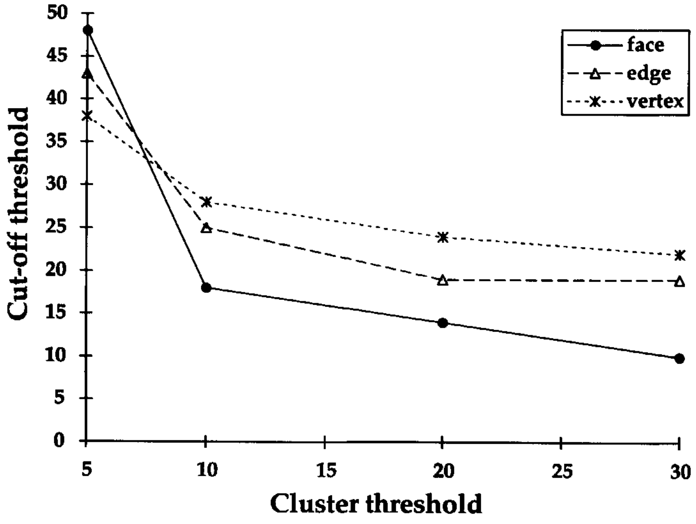


Figure 4.9: The cut-off threshold value as a function of the connected component definition and the cluster threshold.

Finally, the effect of noise on the obtained ellipsoid attributes is investigated with optimal extraction parameters. Noise with $SD = 15.0$ is added to the data, and the face-connected component definition is used in combination with a cluster threshold of 15 points. In Figure 4.10 errors are plotted as a function of the selection threshold value.

The figure clearly shows a large error in position for low selection threshold values. Small selection thresholds lead to many selected points throughout the complete system, which results in a huge cluster filling up the entire system. Also, the error increases for high thresholds, caused by the fact that the feature is small and additional selected points due to noise affect the position significantly. Between the two extremes the errors are stable, and the results of the method are relatively insensitive to noise.

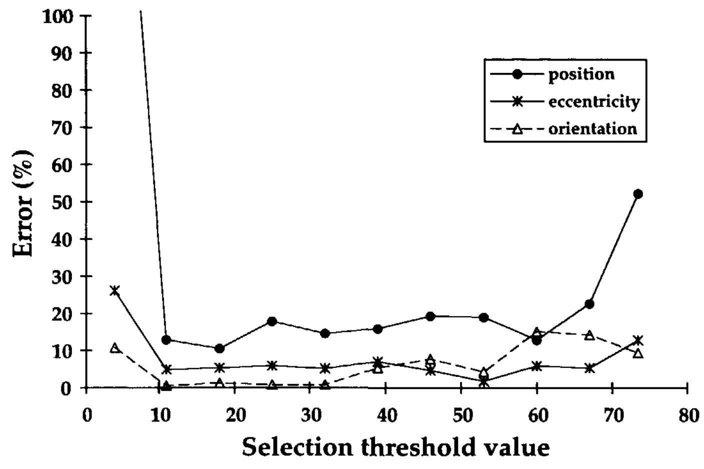


Figure 4.10: The errors of the ellipsoid detection of data with noise ($SD = 15.0$).

4.5 Guidelines for Attribute Calculation

During the execution of the experiments it became clear that much has been learned about the behavior of the feature extraction method. Therefore, we consider this type of experiments extremely important for the exploration and validation of visualization techniques, and we recommend to do similar studies with any new visualization method.

In our case, the following guidelines can be given:

- The cluster threshold should be chosen with great care. A low cluster threshold results in less accurate attributes. A cluster threshold smaller than 5 points results in errors higher than 20% of the grid cell size. Setting the cluster threshold higher than 15 points result in safe attributes with an error below 7% of the grid cell size.

- The cluster threshold is also a powerful tool to eliminate spurious features due to noise. Large cluster thresholds result in correct feature extraction even close to the noise level. This is caused by the spatial coherence of the feature points, while selected points caused by noise are incoherent. However, setting the cluster threshold too high will eliminate small but genuine features.

- The face-connected component amplifies the effects of the cluster threshold of eliminating spurious small clusters. In case of noisy data, this definition is recommended. If noise is absent, the vertex definition is recommended because it results in larger and more significant features.

- If the noise level and distribution (Gaussian) is known in advance, the selection threshold should be set at least $1 * SD$ in combination with a cluster threshold of 20, and $2 * SD$ in combination with a cluster threshold of 5. These combinations guarantee the elimination of all spurious features due to noise.
- The obtained ellipsoid attributes are stable despite the presence of noise. This means that the ellipsoid attributes are relatively insensitive to noise.

The results of the experiments described in this chapter suggests some interesting further studies:

- Small spurious features may be filtered out by morphological operators like opening and closing. This may enhance the effects of the cluster threshold.
- Further statistical analysis can be done on the extraction of features below noise level. Besides coherence in space, coherence in time may be exploited: e.g. if a feature is detected at one time, a prediction can be made of the feature some time later; this prediction can be used to extract the new feature. This suggests a predictive approach for feature tracking in time-dependent data (see Chapter 6).

Chapter 5

Skeleton Shape Description

The preceding chapter discussed two methods for attribute calculation: the volume integrals and the ellipsoid fitting method. Although they result in important and useful attribute sets, they do not provide an accurate description of *shape*. The volume integrals can provide global measures for geometry, like average position, position distribution, and total volume. The ellipsoid fit provides similar information in a more susceptible fashion. It gives a good indication of position, orientation, and size. However, these attributes are a crude (first order) approximation of shape.

For certain applications a more accurate description of the shape of a feature is needed. For instance, the turbulent vortex structures in Section 8.3 are strongly curved 'worms'. Figure 5.1 shows an example of one vortex structure. The segmented region is indicated by an iso-surface, and the ellipsoid attributes are mapped on an ellipsoid icon. The ellipsoid gives a good indication of position and size, but a bad indication of shape and structure of the object. In this case, a more sophisticated description of the geometry is desirable.

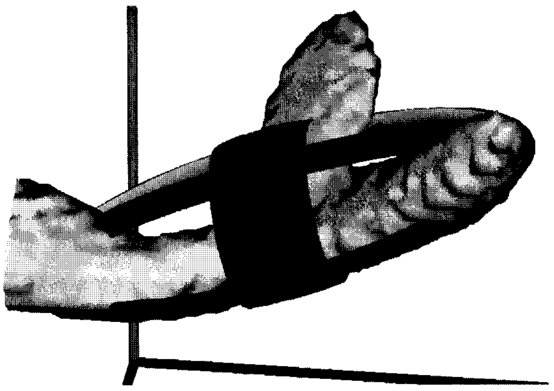


Figure 5.1: An example of a turbulent vortex structure described by an ellipsoidal fit that results in a bad description of shape.

A good description for the shape of an object is provided by the skeleton, or Medial Axis Transform (MAT) [Blum, 1967]. The skeleton of an object can be defined as the locus of points that lie at the center relative to the object's boundary. Thus, the skeleton is a thinner version of a 3D object, which still preserves the basic topology and geometry. Therefore, it is an efficient and compact shape descriptor for objects. Skeletons have been used to describe the shape of features [van Walsum *et al.*, 1996; Gagvani *et al.*, 1998; Puig *et al.*, 2000].

We use the skeleton for the calculation of an attribute set, the skeleton graph, that describes the shape of a feature with a controlled precision (described in [Reinders *et al.*, 2000]). Figure 5.2 illustrates the process for determining the skeleton attributes. Skeletonization determines the voxels[1] on the skeleton of the object. These voxels are then connected into a voxel graph representation with skeleton nodes and edges. Initially, the voxel graph contains all skeleton voxels, but the number of nodes can be reduced by simplification. The result is a skeleton graph attribute set with a smaller number of skeleton nodes and edges that still describes the geometry with a controlled precision.

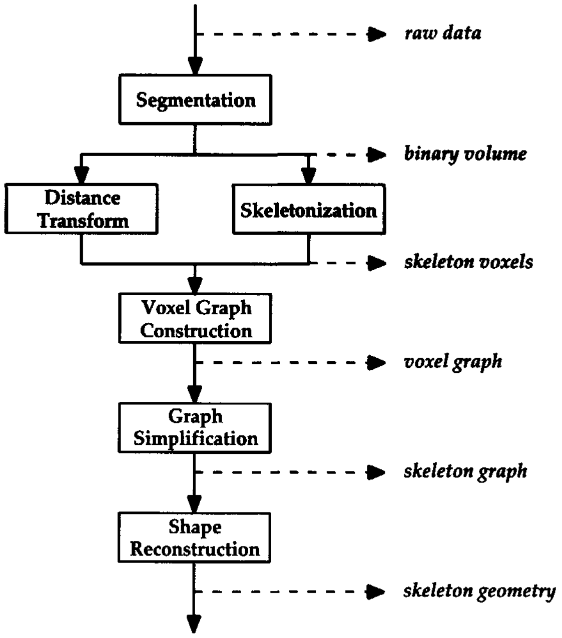


Figure 5.2: Pipeline for the calculation of a skeleton graph attribute set.

Together with the skeletonization process the distance transform (DT) can be calculated for each voxel in the object. The DT is equal to the minimum distance from the voxel to the surface of the object. It can be used during the

[1]Since we use a skeletonization algorithm that works on a regular grid with voxels (the 3D equivalent of pixels), we speak of skeleton voxels instead of skeleton points.

simplification of the voxel graph and during the reconstruction of the shape of the object. Therefore, it is useful to store the DT information at every skeleton node in the skeleton graph.

The skeleton graph can be used to approximately reconstruct the 3D shape of the original object. The skeleton graph is 'fleshed out' by wrapping spherical and conical volumes around the edges and nodes of the graph. The size of these volumes is determined by the DT data. The result is a simplified geometric object, which may be used as an iconic representation describing the feature shape and which provides a more accurate description of shape than the crude approximation by a fitted ellipsoid.

For the steps of skeletonization and distance transform, we used existing algorithms from the image processing literature. We developed our own algorithms for the steps of voxel graph construction, graph simplification, and shape reconstruction.

5.1 Skeletonization

The first step in the skeleton attribute calculation is the skeletonization of the segmented binary volume. The input is a binary volume that is acquired by some segmentation process (see for instance Section 2.3.1). The binary volume tells us if a data point lies inside (TRUE) or outside (FALSE) of the object. Thus, the input can be the result of the selection step in the feature extraction pipeline that we normally use (Section 2.4).

Skeletonization should find the data points that lie at the center of the object, the skeleton points. Many skeletonization algorithms have been published, especially in the image processing literature. Most algorithms are only for 2D data, but some can be extended to 3D or higher dimensions. Roughly, the skeletonization algorithms can be classified in two categories:

- **Topological thinning methods.** These methods iteratively remove all so-called simple points from the surface of the object. These points are points that will not change the topology of the object when they are removed. The points that remain after removing all simple points are the skeleton points that we want to obtain. Examples (in 2D and 3D) of such methods are the *grassfire* methods [Xia, 1989; Leymarie & Levine, 1992a] and methods based on the preservation of the *Euler number* [Lobregt *et al.*, 1980; Lee *et al.*, 1994].

- **Distance transform methods.** The skeleton voxels are identified as the local maxima of the DT. Examples are methods based on *ridge following* [Arcelli & Sanniti di Baja, 1985; Arcelli & Sanniti di Baja, 1989; Xia, 1989; Leymarie & Levine, 1992b]. Another method that uses the DT is presented by [Ogniewicz & Kübler, 1995], which calculates the skeleton from the *Voronoi diagram* of the DT field.

The results of the two types of methods are somewhat different. Topological thinning preserves the connectivity of the skeleton voxels, while DT-methods in general do not. Connectivity between skeleton voxels enables us to reach any skeleton voxel by traversal of neighboring skeleton voxels. For our purposes we want a method that guarantees the connectivity.

The algorithm we use is a topological thinning algorithm[2] based on a hit-or-miss evaluation using sets of masks [Jonker & Vossepoel, 1995]. The mask sets can be used to manipulate a binary object in several ways, of which skeletonization is only one. A mask set consists of a number of $3 \times 3 \times 3$ masks with zeroes, ones and 'don't cares' (only 14 voxels per mask need to be defined because of symmetry reasons). Each mask indicates a configuration that must remain in the object. If the $3 \times 3 \times 3$ neighborhood of a voxel matches one of the masks, it is a skeleton voxel and should remain in the object, otherwise it is removed. Thus, the object surface is peeled off iteratively until only the skeleton voxels remain. Figure 5.3 shows an example of a 3D mask set for an 8-connected skeleton.

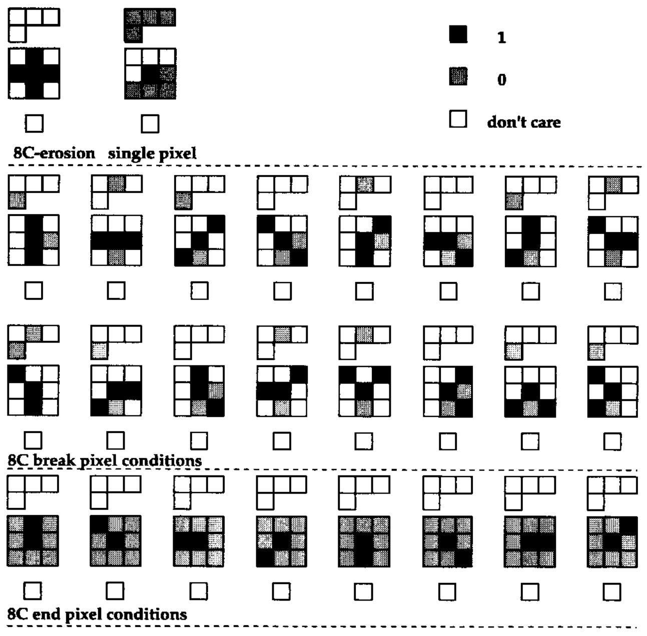


Figure 5.3: A 3D mask set that results in an 8-connected skeleton.

[2]Implementation kindly made available by the Pattern Recognition group at the Applied Physics department of Delft University of Technology.

The characteristics of a skeleton obtained in this way depends on the mask set used. We have three different mask sets to produce different types of skeletons: surface skeletons, line skeletons and point skeletons. Figure 5.4 shows the three different types of skeletons resulting from the same binary volume. The mask set that is used, depends on the type of features that we like to describe: point-type (e.g. critical points), line-type (e.g. vortex cores), or surface-type (e.g. shock waves). The turbulent vortex structures are objects that have a strongly curved, tube-like shape. These are line-type features, so here we concentrate on the line skeletons.

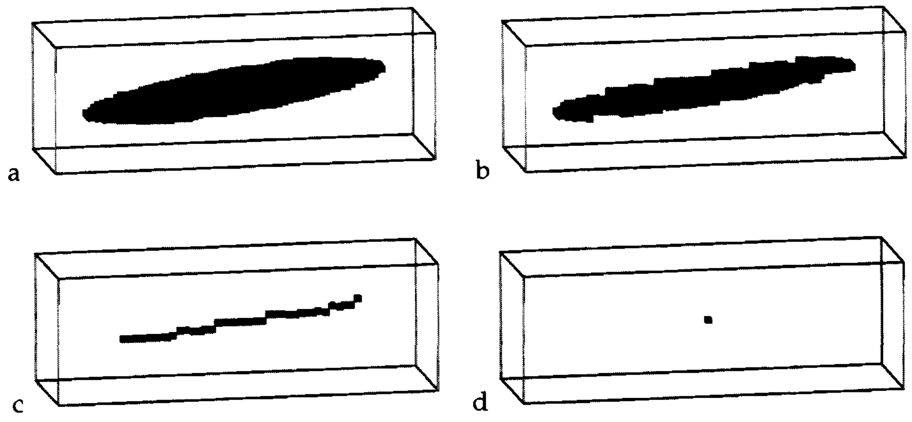


Figure 5.4: Skeletonization of a binary volume results in three types of skeletons depending on the mask set that is used: a) the binary volume, b) surface skeleton, c) line skeleton, and d) point skeleton.

Figure 5.5 shows the skeleton voxels obtained after skeletonization. The object is the same vortex structure as in Figure 5.1. In the figure, the original segmentation is visualized using a transparent iso-surface, and the skeleton voxels are visualized with spheres.

The skeleton voxels follow the structure of the object very well; the topology is well retained. However on the left side, the voxels deviate from the centerline of the object. There, the object is cut by the boundary of the system, causing a flat shape that affects the result. Also, the centerline formed by the skeleton voxels shows wiggles. These are caused by the discrete data representation.

The skeletonization described above is only effective in case of a regular grid where all cells have an equal size in each direction. Each iteration the algorithm peels off one layer of cells from all sides of the surface of the object. In case of an irregular grid the cells on one side of the object may be larger then the cells on the other side. Thus, the object is peeled off in an a-symmetric fashion and

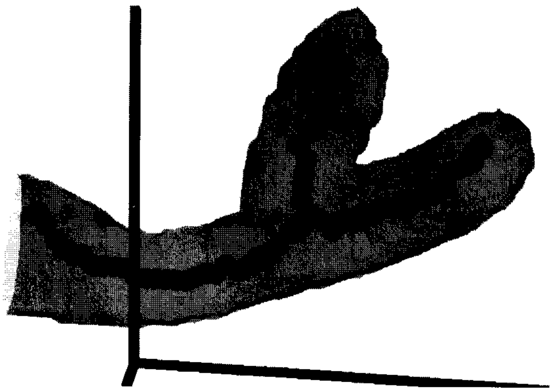


Figure 5.5: The skeleton voxels that result after skeletonization.

the remaining skeleton points will not lie at the geometric center of the object. However, the topology will still be preserved and therefore the skeleton can be useful.

5.2 Distance Transform

Although we do not use a skeletonization method that is based on the DT, it is still useful to calculate the DT. It provides us with distance information; the DT is the minimal distance from a voxel inside the object to the object's surface. The DT can be calculated parallel to the skeletonization process. It is calculated for every voxel in the binary volume, thus it is also available for the skeleton voxels obtained by the skeletonization step.

The DT is well known in the field of mathematical morphology and there are many methods to calculate it [Danielsson, 1980; Borgefors, 1984]. We use a Chamfer distance transformation, which can be determined by two passes as described in [Borgefors, 1984]. The basic idea is that the Euclidean distance between connected neighbors is approximated by weights. Face-connected neighbors get a weight of 10, edge-connected neighbors get a weight of 14, and vertex-connected neighbors a weight of 17. After, the two passes are performed, the values are divided by 10, so that the weights approximate distances of $\sqrt{1}$, $\sqrt{2}$, and $\sqrt{3}$. The result is an approximation of the discrete path length (with wiggles). A better approximation of the Euclidean distance may be obtained by other weights.

5.3 Voxel Graph Construction

After the skeleton voxels have been determined, a *voxel graph* is constructed by connecting neighboring voxels. In the voxel graph, all skeleton voxels are nodes, and adjacent voxels are connected by edges. A skeleton node in the graph contains information about the position of the skeleton voxel, and about the distance to the surface, i.e. the DT. An edge in the graph has two pointers to the two skeleton nodes it connects. The voxel graph is a structure that is easy to manipulate and to analyze. Using the voxel graph, the number of nodes and edges can be reduced, while preserving the basic structure of the skeleton.

The basic approach for constructing the voxel graph is to traverse all voxels of the skeleton and to connect neighboring voxels. However, this is not a straightforward process. Simply connecting all neighboring skeleton voxels causes problems as shown in Figure 5.6 a: a 'zero-loop' originates at the junction. All three voxels at the junction have more than two edges and may be classified as a junction-node (as discussed in Section 5.4.1), while there is in fact only one junction.

The problem of zero-loops is solved in [Lee *et al.*, 1994] by directly classifying the junction node as it is encountered, and classifying all its neighboring nodes as regular. However, the resulting junction node will depend on the order of the traversal, which will result in slightly different graphs, as illustrated in Figure 5.6 b, c, and d.

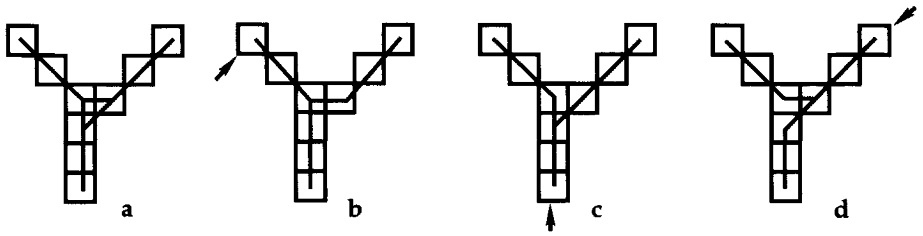


Figure 5.6: Two problems occur when constructing the voxel graph: a zero-loop originates a), and the junction node depends on the traversal starting point b), c), and d).

We solved the problem of zero-loops by assigning a priority ordering to the connections. In a 3D skeleton, each voxel has 26 neighbors: 6 face-connected, 12 edge-connected, and 8 vertex-connected. We give priority to the connection with the nearest neighbor. Thus, face-connected neighbors are connected before edge-connected neighbors and these are connected before vertex-connected neighbors. In case of a junction, this connection priority will always lead to the graph shown in Figure 5.6b, which we believe is the most natural solution.

5.4 Graph Simplification

After the construction of the voxel graph, the graph can be simplified by removing redundant nodes. The voxel graph initially contains all skeleton voxels as nodes, which still may be a large number. We can reduce the number of nodes and edges while still preserving the topology (structure) and geometry (shape) of the skeleton. The task of simplification is to determine which nodes should be kept and which nodes can be removed. There are four reasons for simplification: 1) to reduce size, 2) to remove small insignificant wiggles, 3) to extract topological nodes, and 4) to reconstruct the original shape with simple 3D segments.

There are two classes of nodes that are significant and that should be retained: *topological* nodes and *geometric* nodes. Topological nodes are nodes that are necessary to preserve the topology, and geometric nodes are necessary to preserve, to a certain extent, the basic geometric shape of the object. The identification of topological nodes and their connectivity results in a topological graph, while the detection of geometric nodes results in the geometric graph. The final skeleton graph is a combination of these two graphs.

5.4.1 Topological graph

The topological graph holds the basic structure of the object and can be determined by finding the topological nodes. The topological nodes are identified by counting the number of edges connected to a node in the voxel graph:

- *End node*: one edge.
- *Regular node*: two edges.
- *Junction node*: three or more edges.

The end nodes and junction nodes are the nodes that determine the topology of the object. Thus, the topological graph is created by simply removing the regular nodes, and connecting the remaining nodes by edges.

The topological graph describes the structure of the object. Figure 5.7 shows the topological graph in combination with the original segmented object. The nodes in the graph are shown by spheres and the edges are illustrated by lines. The figure tells us that we have three end nodes and one junction, and it shows how these are connected.

However, there is a problem in case of loops. An additional type of topological node, the loop node, is needed to describe loops in the graph. We define a loop node as a node in the topological graph that is connected twice by the same edge, i.e. it has an edge that connects to itself:

- *Loop node*: has an edge that connects to itself.

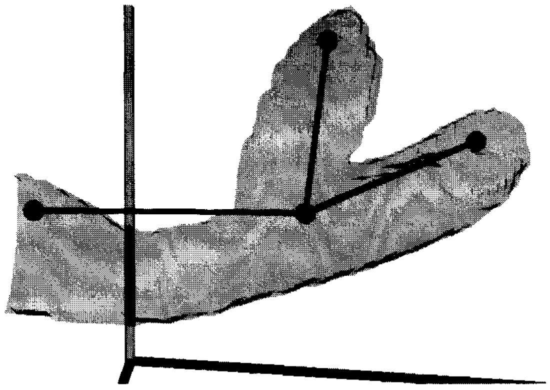


Figure 5.7: The topological graph that describes the structure of the object.

Figure 5.8 shows a number of situations with loops in the graph. In case of Figure 5.8a, the removal of all regular nodes will remove all nodes. To overcome this problem, a loop node is included at an arbitrary location on the loop. In the topological graph the loop node is connected to itself with an edge. A similar situation occurs in Figure 5.8b; the junction node is connected to itself. Therefore, this junction node is classified as a loop node. In the Figures 5.8c and 5.8d, also a loop exists, but the nodes are not connected to themselves, therefore all nodes in the loop remain junction nodes.

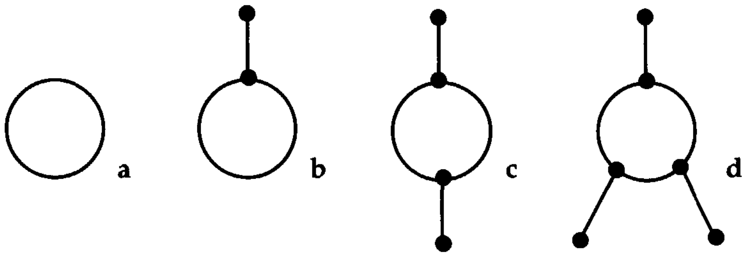


Figure 5.8: Loops in the topology of the skeleton.

The number of loops is an important variable in the topological graph, because it may provide a clue for comparison of two skeletons. The number of loops can be calculated with the *Euler formula*:

$$V - E + F = C - H \tag{5.1}$$

with V the number of vertices or nodes, E the number of edges and F the number of faces. $C - H$ form the so-called *Euler number*, with H the number of holes in an object and C the number of connected objects in a scene. Because we only work with lines F can be set to $F = 0$, and because the graph is always a single object, C can be set to $C = 1$. Substitution gives the number of loops (holes) in the graph:

$$H = 1 - V + E \tag{5.2}$$

This equation is used in Section 7.5 for the detection of topological events.

5.4.2 Geometric graph

The topological graph describes the structure of the object very well, but does not accurately follow the centerline (see Figure 5.7). The geometric graph refines the edges in the topological graph by inserting geometric nodes.

The underlying model for the geometric graph is that the shape of an object can be modeled by a truncated cone representation: conical volumes wrapped around the edges of the graph approximate the object's volume with controlled precision. The centerline of the cones should follow the line of skeleton voxels, and the radius of the cone at points on the centerline should follow the DT at the corresponding skeleton voxels. Thus, the model includes the centerline and the profile of the object.

Geometric nodes are key points that are inserted in order to refine the geometric graph. The geometry can vary in two ways: the skeleton line is curved, or the profile of the object surface varies, i.e. variations in the diameter of the cross-section. Hence, we distinguish the following two types of geometric nodes:

- *Curve nodes*: where the skeleton line bends.
- *Profile nodes*: where the surface profile changes.

The geometric nodes are inserted on the edges of the topological graph. For each edge the regular nodes, between the two end nodes of the edge, are traversed, and curve or profile nodes are inserted. Insertion of geometric nodes depends on geometric tests, which will be described below.

Curve nodes

Curve nodes are found by testing the distance between the skeleton voxel node positions and the line between two end nodes of the corresponding edge in the skeleton graph. The maximum distance d_{max} is determined and compared to a curve threshold T_{curve}; see Figure 5.9. A curve node is added at the location of the maximum when $d_{max} > T_{curve}$. Two new edges are created and both are tested recursively. The process terminates when all intermediate voxel nodes fall within T_{curve} of the corresponding edges.

The curve threshold T_{curve} provides a measure of precision for the approximation of the centerline of the original voxel graph. A zero threshold results in

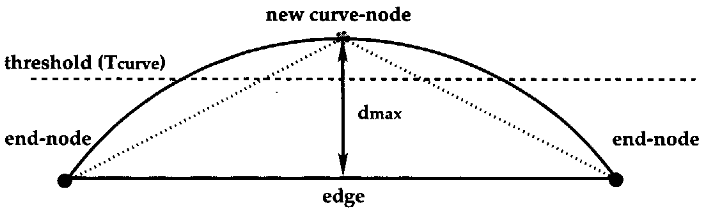


Figure 5.9: Finding the curve nodes recursively.

a geometric graph that is identical to the original voxel graph. When the threshold is very large, no curve nodes are inserted and the topological graph is not refined.

Profile nodes

Profile nodes are found by comparing the DT at the skeleton nodes to the radius of the cone wrapped around the edge. The two radii at the two ends of the cone are equal to the DT of the corresponding nodes, and the profile of the cone is linear along the edge. However, the DT values of the intermediate skeleton voxels will differ from this linear profile. The test is similar to the test for finding the curve nodes. The profile test finds the voxel with the maximum distance between its DT and the linear profile and tests this distance to a profile threshold T_{prof} (see Figure 5.10). A new profile node is inserted when the threshold is exceeded and the two resulting edges are tested recursively.

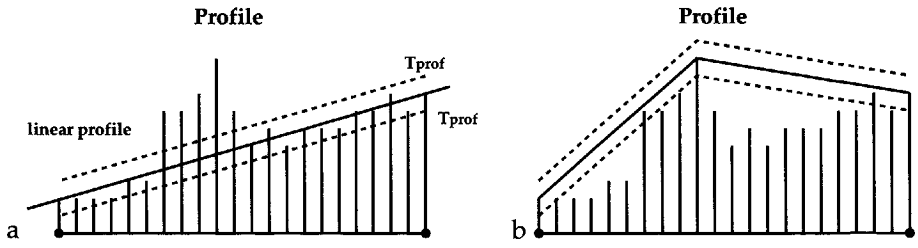


Figure 5.10: Finding the profile nodes recursively: a) first step, and b) second step.

The threshold is a measure for shape approximation of the original voxel graph, but this time for the profile of the surface of the object. Again, with a zero threshold the geometric graph is equal to the voxel graph, and with a very large threshold the topological graph remains unchanged.

The geometric graph is a simplified replica of the voxel graph where the precision of the reproduction can be controlled with the thresholds T_{curve} and T_{prof} (typical values are $T_{curve} = T_{prof} = 2.0$). Typically, the number of nodes can be reduced by a factor in the order of 10. The reduction factor depends on the application (the curvature of the features) and on the chosen thresholds. Figure 5.11 shows the skeleton voxels by small boxes and the skeleton graph is shown by spheres connected by lines. The skeleton graph edges closely follow the centerline of skeleton voxels.

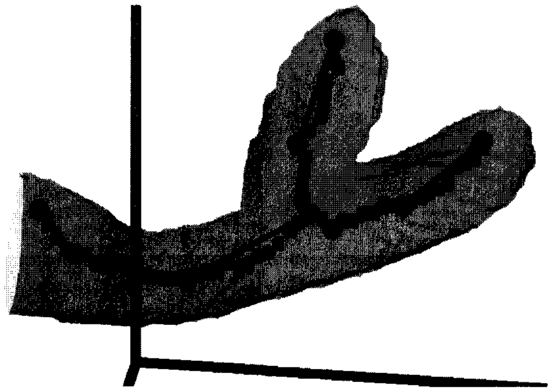


Figure 5.11: The skeleton graph is a good approximation of the voxel graph. The precision can be controlled by the thresholds T_{curve} and T_{prof}.

Figure 5.12 shows the skeleton graph with nodes shown by spheres that have a radius equal to the DT. In general, the skeleton graph follows the thickness of the object well. However, it is less accurate when the object has a flat shape, as in case of the upward branch. The radii of the spheres seem too small for this part. The upward branch has a flat shape, with a cross-section that has an elliptical contour. We assume a circular contour with a radius equal to the DT, which is a minimal distance to the surface.

5.5 Shape Reconstruction

The shape of the original object can be reconstructed using the skeleton graph. An approximation can be made by 'fleshing out' every edge in the skeleton graph, i.e. by adding a volume to every edge. One way is to calculate a cubic Hermite interpolation tube [van Walsum *et al.*, 1996] through each path (from end node to end node) in the skeleton graph. The interpolation results in a smooth tube, as shown in Figure 5.13. The thickness of the tube is equal to the distance information at the nodes. This way, a smooth surface is drawn representing the feature objects.

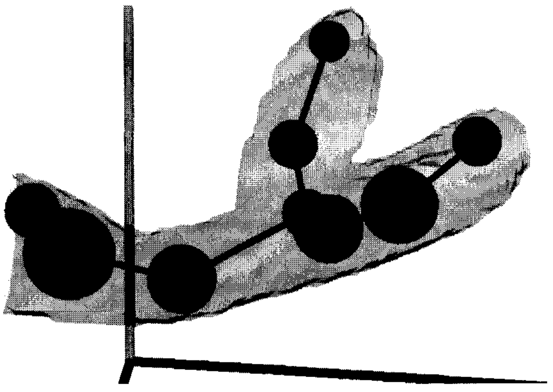


Figure 5.12: A visualization of the final skeleton graph. The nodes and edges are shown by spheres and lines, and the surface of the object is shown transparently.

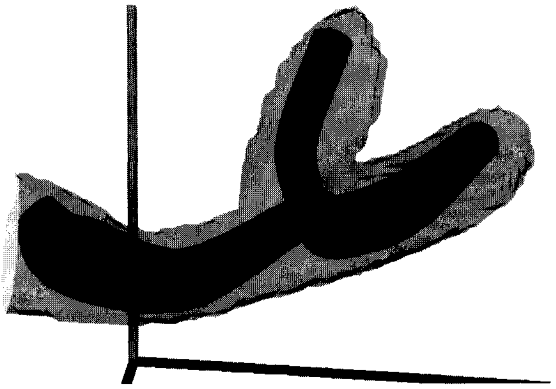


Figure 5.13: Skeleton shape reconstruction using Hermite tube interpolation.

The Hermite tubes provide a smooth reconstruction of the shape, but they are expensive since they require many polygons to be rendered. This is fine when the shape reconstruction by itself was our goal, but unfavorable when a fast display is required, for instance when the evolution in time of the features is visualized by displaying the shape reconstruction frame by frame.

A less expensive way to visualize the skeletons is the use of simple icons [van Walsum *et al.*, 1996]. Several types of geometric icons can be used to visualize the edges in the skeleton graph. Figure 5.14 shows a number of icons that visualize an edge. Figure 5.14a is a combination of two spheres and a truncated cone. The spheres represent the two skeleton nodes of the edge, and the cone is the volume fitted to the edge. The radii of the spheres and cone are set according to the DT known at that node.

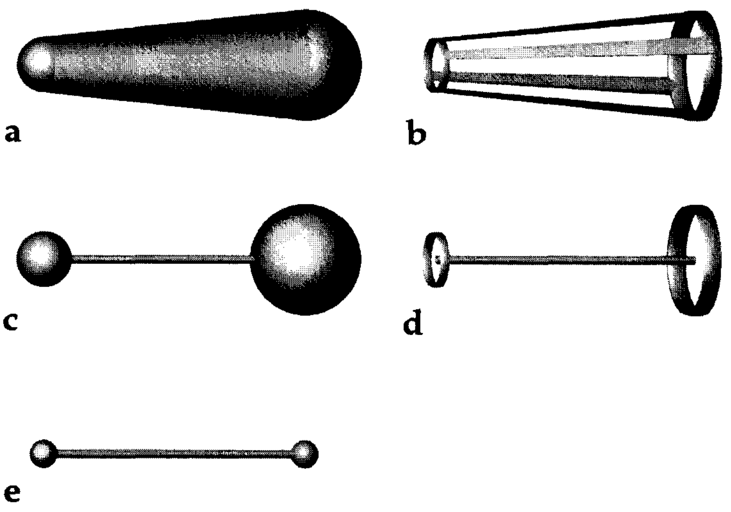


Figure 5.14: Icons that can be used to visualize an edge of the skeleton graph.

Figures 5.14b, c, d, and e show more open geometries. This reduces the effects of cluttering and gives more focus to the topology rather than geometry. Figure 5.14e does not use the distance information at all and is especially suited when topology is important.

By drawing all nodes and edges in the skeleton graph, a complete representation of the skeleton is obtained. Figure 5.15 shows the skeleton graphs obtained for one frame in the application of the turbulent vortex structures (Section 8.3). Using the spheres and cone representation for an edge, Figure 5.15a, a solid shape reconstruction is obtained (this figure is shown in color on the backside of the cover). Figure 5.15b, c, and d show the other, more open iconic representations.

5.6 Accuracy of the Skeleton Attributes

We will now investigate the accuracy of the skeleton attributes in a similar fashion as the accuracy tests in Chapter 4. Here, we test the accuracy of two derived quantities from the skeleton graph: the average position and total volume of the skeleton. As a reference we use the position and volume obtained by the volume integral which proved to be very accurate and stable in Chapter 4.

A number of derived quantities can be calculated from the skeleton graph de-

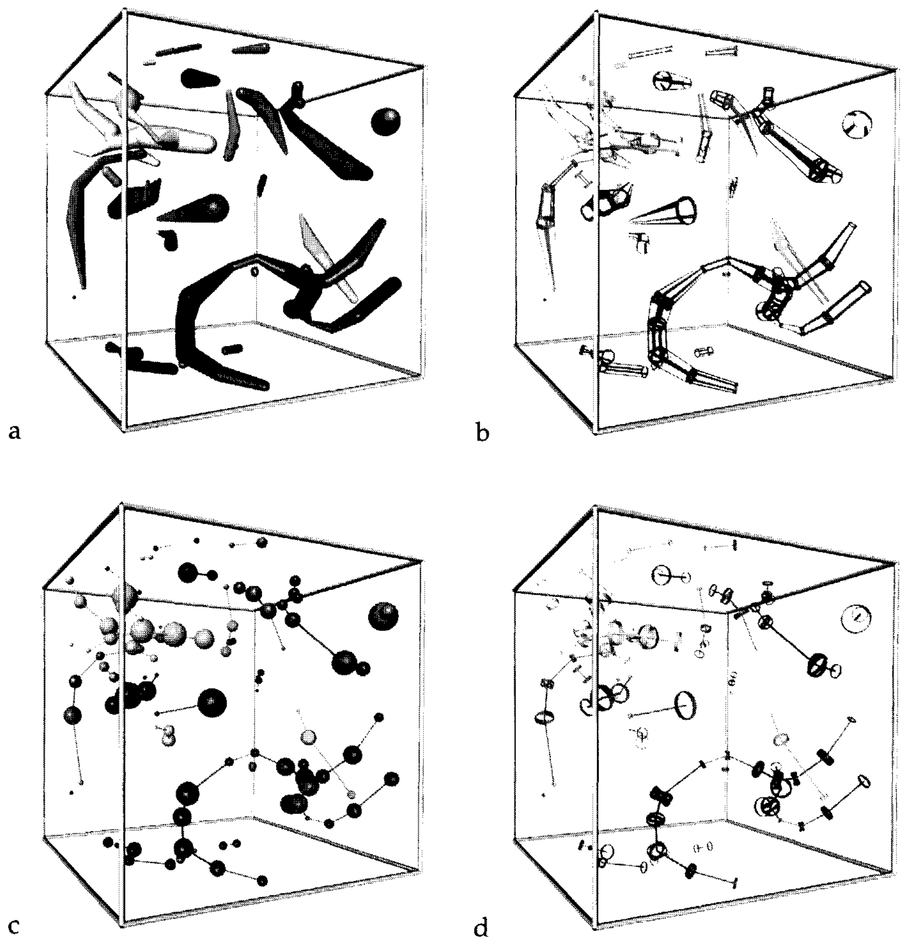


Figure 5.15: Iconic visualizations of skeletons using spheres and cones, or more open iconic representations, a) is shown in color on the backside of the cover.

scribing global attributes of the skeleton: the length of the skeleton centerline L_{sk}, the total skeleton volume V_{sk}, and the skeleton position of the center of gravity (COG) $\mathbf{P}_{sk}$. These quantities are important since they can be used for comparison and for tracking, see Section 6.2.1.

1. The length of the skeleton centerline L_{sk} can be calculated as a sum of the edge lengths over all N edges:

$$\begin{aligned} L_{sk} &= \sum_{i}^{N} L_i \\ \text{with} & \\ L_i &= \left\|\mathbf{P}_{i,2} - \mathbf{P}_{i,1}\right\| \end{aligned} \tag{5.3}$$

where L_i is the distance between the two node positions $\mathbf{P}_{i,1}$ and $\mathbf{P}_{i,2}$ of edge i.

2. The total volume of the skeleton V_{sk} is the sum of the volumes over all edges and end nodes. The volume of an edge V_e is approximated by the volume of a truncated cone with a top-radius of DT_1 and a bottom radius of DT_2. The volume of an end node V_n is equal to half the volume of a sphere (the other half is contained in the cone representing the edge starting from that end node). The volume can be approximated by a sum over all N edges plus a sum over all M end nodes:

$$\begin{aligned} V_{sk} &= \sum_{i}^{N} V_{e,i} + \sum_{j}^{M} V_{n,j} \\ \text{with} & \\ V_{e,i} &= \frac{1}{3}\pi L_i \left(DT_{i,2}^2 + DT_{i,2}DT_{i,1} + DT_{i,1}^2\right) \\ V_{n,j} &= \frac{2}{3}\pi (DT_{n,j})^3 \end{aligned} \tag{5.4}$$

where $V_{e,i}$ is the volume of edge i, $V_{n,j}$ is the volume contribution of end node j, and $DT_{n,j}$ is the distance transform information at end node j.

3. The position of the COG of the skeleton $\mathbf{P}_{sk}$ is the weighted sum of COG's over all edges and end nodes. The COG of an edge $\mathbf{P}_e$ is equal to the COG of a truncated cone. A halfway factor H_f is calculated that determines the right position, $H_f = 0.5$ when $DT_1 = DT_2$. The skeleton position is calculated by a sum over all N edges and M end nodes adding their positions weighted by their volume:

$$\mathbf{P}_{sk} = \frac{1}{V_{sk}} \left(\sum_{i}^{N} V_{e,i}\mathbf{P}_{e,i} + \sum_{j}^{M} V_{n,j}\mathbf{P}_{n,j} \right) \tag{5.5}$$

$$\begin{aligned}
&\text{with} \\
\mathbf{P}_{e,i} &= \mathbf{P}_{i,1} + \left(\mathbf{P}_{i,2} - \mathbf{P}_{i,1}\right) H_f \\
&\text{with} \\
H_f &= \left(\frac{1/2\left(4DT_2^3 + 4DT_1^3\right)^{1/3} - DT_1}{DT_2 - DT_1}\right) \\
&= 0.5 \text{ when } DT_1 == DT_2
\end{aligned}$$

where $\mathbf{P}_{e,i}$ is the position of the COG of the edge i, and $\mathbf{P}_{n,j}$ is the position of the end node j.

The first accuracy test uses synthetic data generated by the procedure described in Section 4.3.1. We have tested the influence of the shape of a feature on the volume reconstruction of the skeleton graph description of the feature. For this test, we have generated a number of ellipsoid-shaped features, with different axis ratios. We can recognize three extreme cases:

1. a 'cigar'-shaped ellipsoid (axis $R_1 = R_2 \ll R_3$)
2. a 'sphere'-shaped ellipsoid (axis $R_1 = R_2 = R_3$)
3. a 'disc'-shaped ellipsoid (axis $R_1 = R_2 \gg R_3$)

We expect that our skeleton graph can accurately represent line-type (cigar) and point-type (sphere) features, but has problems with surface-type (disc) features. A synthetic feature is slowly changed from cigar to sphere and from sphere to disc. At each step, the skeleton graph is determined and the derived volume is compared to the volume obtained by a volume integral.

Figure 5.16 shows the results of our tests. On the y-axis is the ratio of the skeleton volume and the integral volume, which should ideally be 1. The x-axis plots the ratio of the axes:

$$\frac{R_1 + R_2}{2R_3} \tag{5.6}$$

In the middle is the sphere, with ratio 1. To the left the ratio is < 1, i.e. the cigar shape, and to the right the ratio is > 1, i.e. the disc shape.

Our expectations are confirmed: flat objects are not represented very well. The volume ratio decreases very quickly on the right side of the graph. The left side of the graph shows that the profile threshold has a significant effect. This can be explained as follows: the cigar-shaped ellipsoid has a straight centerline with small DT values at the end nodes, curve nodes will not be inserted, but one profile node in the middle will have a large effect on the skeleton volume. Hence, the stepwise behavior of the graph: at each step a new profile node is inserted.

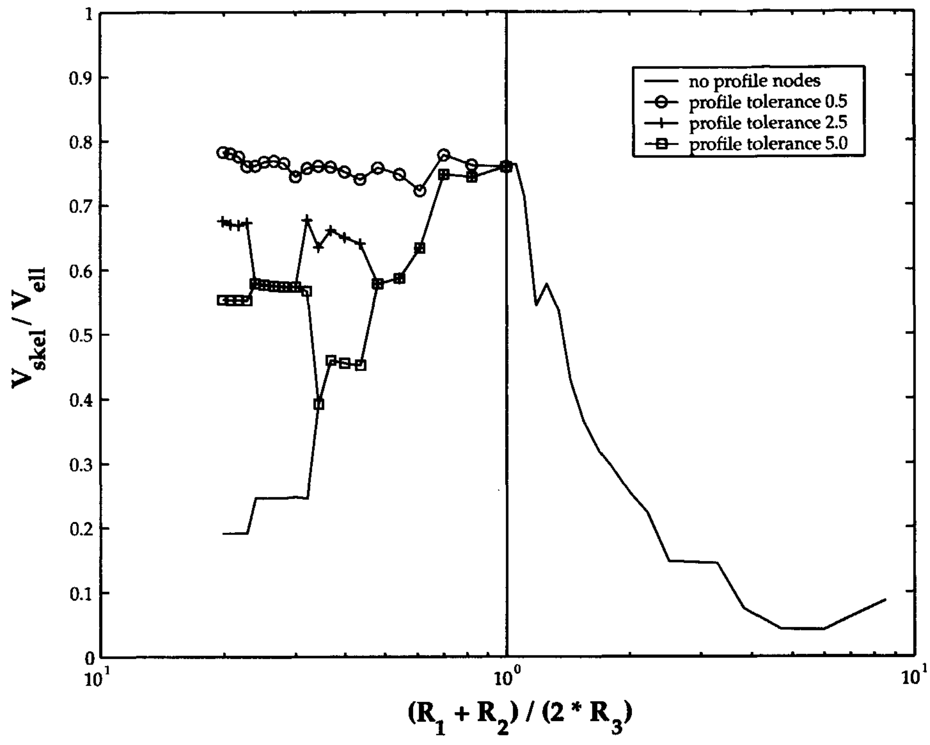


Figure 5.16: The skeleton volume ratio for different axis ratios.

A second test is performed using the turbulent vortex structures application discussed in Section 8.3. We tested the skeleton attributes with the volume integral attributes of every vortex structure in the 100 frames. The skeleton graph COG position has an average distance of 1.4 voxels to the volume integral position. This is relatively small compared to the system dimensions of 128^3.

The average volume ratio for all features is about 42%. The volume ratio seems to be an unfavorable fraction, but this is a bit exaggerated because the volume is proportional to the square of the distance DT. The problem lies in the fact that the distance DT is the minimal distance from each skeleton point to the surface of the object. A possible solution to that problem will be discussed in the next section. For now, we can only conclude that we need to be careful, when using the skeleton volume attributes for further analysis.

5.7 Future Prospects

The skeleton graph can provide a good shape reconstruction for line-type features like the turbulent vortex example shown before. However, it is also clear that surface-type features are approximated with lower accuracy. The assumption that the contour is a circle is in general not justified. Moreover, the DT is the minimal distance to the surface which results in an even worse approximation, illustrated in Figure 5.17a.

The contour may be approximated better by taking the average distance D_{avg} from the contour points to the skeleton node 5.17b, by determining an ellipse fit around the contour 5.17c, or by sampling the contour along radial lines emanating from the skeleton node 5.17d. The latter method is similar to the way the contour of a vortex core is constructed in [Banks & Singer, 1995].

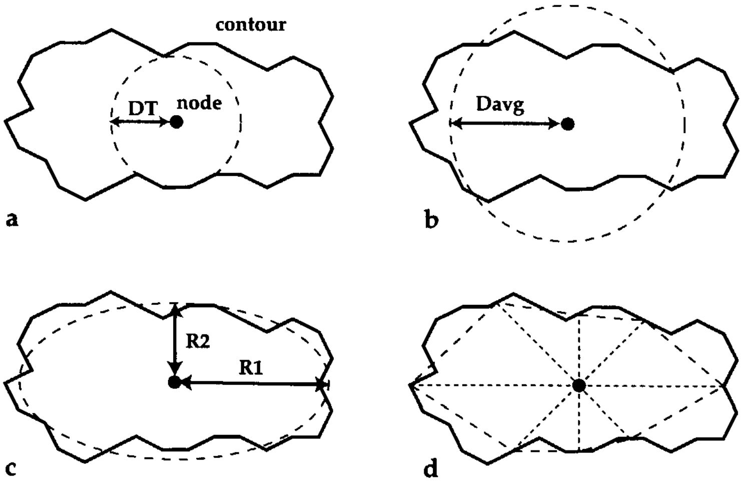


Figure 5.17: Contour approximation by a) DT, b) average distance, c) ellipse fitting, and d) sampling along radial lines.

The three solutions given above provide more precise surface approximations, however they also require two extra steps. First, the contour in a plane perpendicular to the skeleton line has to be determined. Second, the contour has to be approximated. These two steps may involve many calculations and are thus computationally more expensive.

Another issue for future work is the description of surface-type features, such as shock waves. Surface-type features may be described using the mask set that results in a surface skeleton during the skeletonization step, see Figure 5.4b.

The skeleton voxels can be connected by triangles, the result is a polygon mesh description of the skeleton surface. This surface can be simplified by a surface simplification method [Schroeder *et al.*, 1992; Klein *et al.*, 1996; Cignoni *et al.*, 1998]. In [Frank & Lang, 1998], a data-dependent component is taken into account during simplification. The DT can be taken as the data-dependent component. In that way the surface is simplified by both the curvature of the triangle mesh and the distance information stored at each skeleton point. This should provide controlled precision in a similar fashion as the graph simplification we use.

Chapter 6

Feature Tracking

The preceding two chapters discussed a number of methods for the calculation of attribute sets that describe certain characteristics of features. These attribute sets can be stored in the feature data representation. The feature data representation allows the manipulation of features as individual entities and the features can be quantitatively compared. Thus, the features can be investigated further, for instance to investigate their evolution in time.

The feature data representation can be used for the tracking of features in time-dependent data. Feature tracking is one step in the process of the visualization of time-dependent phenomena (see Section 2.5). The process consists of feature extraction, feature tracking, event detection, and visualization (Figure 2.11). The sequence as a whole can be described: for each feature a description can be made of its lifespan, describing its origin, motion, growth, interaction with other features, and so forth.

This chapter describes an algorithm for the task of feature tracking using feature data only. The use of a feature data representation has important consequences: the features can be quantitatively compared without using the original grid data. Now, it is feasible to perform the tracking interactively and in multiple passes. The chapter also presents ways to visualize the results of feature tracking, and compares our method to other methods.

6.1 Correspondence Problem

Feature tracking tries to solve the frame-to-frame *correspondence problem* between features in successive frames. The correspondence problem originates from the fact that the feature extraction step is performed independently for every frame. The features in one frame are not related to the features in the next frame. It is not yet known if two features in successive frames are actually the same object[1] at two different instances of time. Feature tracking tries to solve this correspondence problem.

Feature tracking results in one-to-one correspondences between features in subsequent frames, i.e. only *continuations* are detected (see Section 7.2). Here we

[1]We prefer to speak of an 'object' when a feature is tracked and its evolution in time is known.

presume that during its lifespan, the object continues to exist without dissipation or interaction with others. The feature descriptions in every time-step are stored in an *object path* describing the continuous evolution of the object. Thus, tracking allows us to select one object path and visualize the evolution of it.

The continuous object paths are the basis for event detection (Chapter 7): they can be used to detect certain events such as unusual changes, particular stages in the evolution, or specific interactions with other features.

When displaying features frame by frame, the eye sees visual correspondence based on continuity of moving objects, and is very flexible in tracking objects in time. However, it is difficult to translate this flexibility into an algorithm that can be used for feature tracking. Chapter 7 of [Ballard & Brown, 1982] describes a number of heuristics for the visual matching of points in successive images separated by a small time-interval:

1. Maximum velocity. The motion of a point in the image is limited according to a maximum velocity.
2. Small velocity change. The velocity of a point in the image changes little to the next time-step.
3. Common motion. Spatially coherently moving objects often appear in successive images as regions of points sharing a 'common motion'.
4. Consistent match. Two points from one image generally do not match a single point from another image (exceptions arise from occlusions).
5. Known motion. If a world model can supply information about object motions, perhaps motions can be derived, predicted, and recognized.

We will try to translate these heuristics to rules used in our feature tracking algorithm.

The tracking method described here tries to combine the amenities of the methods described in Section 2.6.2. The method uses *continuity of motion* to track paths, and *tolerances* to allow small deviations.

6.2 Feature Correspondence

The first step in feature tracking is to create a measure for the correspondence between two features F_1 and F_2. We use the feature data to make a quantitative comparison. The difference between the attributes in both feature descriptions is tested with a certain tolerance. The tolerance allows a deviation between the attributes in F_1 and F_2. The correspondence criteria are similar to the criteria described in [Samtaney *et al.*, 1994].

6.2.1 Correspondence functions

Each attribute set in the feature data is associated to a number of correspondence criteria. The criteria depend on the information available in that attribute set. An ellipsoid fitting attribute set is linked to other correspondence criteria than a skeleton attribute set. Basically, the criteria have the following format:

$$func(F_1, F_2) \leq T_{func} \tag{6.1}$$

where $func(F_1, F_2)$ is some comparison function that compares an attribute set in F_1 with an attribute set of the same type in F_2, and T_{func} is the function tolerance. Each type of attribute set can return a list of comparison functions $func$ that can be tested to a tolerance.

Each correspondence criterion is translated in a correspondence function C_{func} by equation 6.2. It has the following properties: $C_{func}(F_1, F_2) = 1$ when the attributes in the two features are identical, $C_{func}(F_1, F_2) = 0$ when the deviation is on the limit of the tolerance, and $C_{func}(F_1, F_2) \leq 0$ when the deviation exceeds the tolerance.

$$\begin{aligned} C_{func}(F_1, F_2) &= 1 - \frac{func(F_1, F_2)}{T_{func}} \\ &= \begin{cases} 1 & \text{Exact match} \\ 0 & \text{Limit tolerance} \\ < 0 & \text{Exceed tolerance} \end{cases} \end{aligned} \tag{6.2}$$

Hence, a correspondence between F_1 and F_2 is found when the correspondence function returns a positive value. Since for one attribute set there may be more than one correspondence criterion, the possible correspondence functions are combined into a correspondence factor as described in Section 6.2.2.

The correspondence criteria depend on the information in the attribute set, and on the different types of attributes in that attribute set. Table 6.1 shows a number of possible correspondence criteria for basic types of attributes, such as scalar, vector, and matrix attributes. These criteria are templates for the criteria that can be used in specific attribute sets. However, each attribute set may add specific criteria that are only meaningful for that particular attribute set.

Most of the correspondence criteria in Table 6.1 provide a relative measure for the difference in attribute values. Equation 6.3 expresses the relative difference between two scalar values relative to the maximum of the two values. It can be used for several scalar attribute types such as volume, mass, and average data values. It is an important measure because it is also used as a template for vector and matrix-type attributes. For instance equations 6.4, 6.5, and 6.8 respectively give the relative distance between two vectors, the relative difference in vector length and the relative difference of the matrix determinant. The accompanying tolerance value T_{func} is also relative, i.e. a tolerance value of $T_{func} = 0.3$ indicates that a relative difference of 30% is allowed.

Attribute type	Correspondence criteria
Scalar S	$\frac{\lvert S_2 - S_1\rvert}{\max(S_1, S_2)} \leq T_{scalar}$ (6.3)
Vector $\mathbf{V}$	$\frac{\lVert \mathbf{V}_2 - \mathbf{V}_1 \rVert}{\max(\lVert \mathbf{V}_1 \rVert, \lVert \mathbf{V}_2 \rVert)} \leq T_{diff}$ (6.4) $\frac{\lvert \lVert \mathbf{V}_2 \rVert - \lVert \mathbf{V}_1 \rVert \rvert}{\max(\lVert \mathbf{V}_1 \rVert, \lVert \mathbf{V}_2 \rVert)} \leq T_{length}$ (6.5) $1 - \left\lvert \frac{\mathbf{V}_1}{\lVert \mathbf{V}_1 \rVert} \cdot \frac{\mathbf{V}_2}{\lVert \mathbf{V}_2 \rVert} \right\rvert \leq T_{angle}$ (6.6) $\lVert \mathbf{V}_2 - \mathbf{V}_1 \rVert \leq T_{dist}$ (6.7)
Matrix M	$\frac{\lvert \det(M_2) - \det(M_1)\rvert}{\max(\det(M_1), \det(M_2))} \leq T_{det}$ (6.8)

Table 6.1: Possible correspondence criteria for basic types of attributes.

Sometimes the relative difference measure is not an appropriate measure. For instance, in case of a position attribute $\mathbf{P}$ equations 6.4 and 6.5 are meaningless because position depends on the coordinate system that is used and thus $\max(\lVert \mathbf{P}_1 \rVert, \lVert \mathbf{P}_2 \rVert)$ is arbitrary. In this case, it is better to use an absolute measure like in equation 6.7. The measure is the absolute distance between the two positions. Hence, the tolerance T_{pos} is also absolute and one should take care to choose a tolerance that makes sense for this particular case, i.e. it depends on the changes in the motion of the objects, and the spatial dimensions of the system.

The useful correspondence criteria depend on the application and the meaning of the attributes that were calculated. For example, suppose $\mathbf{V}$ is the average velocity of the flow in a region of interest in a CFD data set, then equations 6.4, 6.5, and 6.6 are useful correspondence criteria. Equation 6.4 calculates the relative difference between the two velocity vectors. Equation 6.5 gives the relative difference in speed. Equation 6.6 uses the in-product of the two normalized velocity vectors to give a measure for the difference in velocity direction. The example shows that not all possible correspondence criteria make sense. It depends on the attribute set and on the application which criteria are actually useful. Below we will discuss the correspondence criteria that are associated to the ellipsoid and skeleton attribute sets.

The correspondence criteria associated to other types of attribute sets may be derived in a similar fashion as the correspondence criteria associated to the ellipsoid and skeleton attribute set. Each type of attribute set returns a number of criteria that test the information available by that particular attribute set. Thus, for feature correspondence all relevant information in the attribute sets can be used.

Ellipsoid correspondence functions

The above correspondence criteria serve as a template for the criteria that are associated to the ellipsoid fitting attribute set. Ellipsoid data consists of the position, the three axis lengths, and the three rotational angles. The information that is held in these attributes concern position, size, and orientation. Hence, the correspondence criteria listed in Table 6.2 involve these three types of information.

Position	$\|\mathbf{P}_2 - \mathbf{P}_1\| \leq T_{pos}$	(6.9)
Volume	$\frac{\lvert V_2 - V_1 \rvert}{\max(V_1, V_2)} \leq T_{vol}$	(6.10)
Orientation	$1 - \left\lvert \frac{\mathbf{R}_1}{\|\mathbf{R}_1\|} \cdot \frac{\mathbf{R}_2}{\|\mathbf{R}_2\|} \right\rvert \leq T_{angle}$	(6.11)

Table 6.2: Correspondence criteria associated to the ellipsoid fitting attribute set, $\mathbf{P}$ = position, V = volume, and $\mathbf{R}$ = the vector of the main axis.

Equation 6.9 tests the distance between the two center positions of the ellipsoids. Equation 6.10 tests the relative difference in volume of the two ellipsoids. Equation 6.11 tests the angle between the main axes of the two ellipsoids. This test is only meaningful when the main axis is clearly longer than the other two axes and has a particular significance in the application.

Skeleton correspondence functions

The correspondence criteria linked to the skeleton attribute set mainly concern derived quantities. In Section 5.6, we describe the calculation of a number of global quantities derived from the skeleton graph: the skeleton position of the center of gravity $\mathbf{P}$, the skeleton volume V, and the skeleton length L.

The correspondence criteria associated to the skeleton attribute set are listed in Table 6.3. The tests concerning the global quantities of position and volume (equations 6.12 and 6.13) are similar to the tests associated to the same quantities of the ellipsoid attribute set. However, the quantities are calculated differently and thus the tracking based on these tests may be different.

Position	$\|\mathbf{P}_2 - \mathbf{P}_1\| \leq T_{pos}$	(6.12)
Volume	$\frac{\lvert V_2 - V_1\rvert}{\max(V_1, V_2)} \leq T_{vol}$	(6.13)
Length	$\frac{\lvert L_2 - L_1\rvert}{\max(L_1, L_2)} \leq T_{length}$	(6.14)
Topology	$\lvert TE_2 - TE_1\rvert + \lvert NL_2 - NL_1\rvert \leq T_{topo}$	(6.15)

Table 6.3: Correspondence criteria associated with the skeleton attribute set, $\mathbf{P}$ = position, V = volume, L = length, TE = the number of edges in the topological graph, and NL = the number of loop nodes according to the Euler formula, see equation 5.2.

The criteria in equations 6.14 and 6.15 are typical criteria for the skeleton attribute set. Equation 6.14 tests the skeleton length and equation 6.15 tests the number of topological differences of the two skeleton graphs. The topology of two skeletons can also be tested in other ways, but that will be discussed in Section 7.5.

The correspondence criteria are similar to the criteria of [Samtaney *et al.*, 1994], however, the present approach is much more powerful. The use of our feature data representation allows us to choose from many attribute sets and decide which attributes are relevant for this particular application. The correspondence criteria related to the attributes are automatically shown in the interface, as illustrated in Section 6.6. While [Samtaney *et al.*, 1994] always use the same set of attributes and if this set is inadequate, certain tracking procedures must be rewritten. In our case, we only have to create a new subclass of attribute set, and write the correspondence functions associated to it. The tracking process itself does not need to be changed.

6.2.2 Correspondence factor

The correspondence between features F_1 and F_2 can be calculated by combining the correspondence functions into one *correspondence factor*. The correspondence factor is calculated as follows:

$$Corr(F_1, F_2) = \frac{\sum_{i=1}^{N_{func}} C_i(F_1, F_2) * W_i}{\sum_{i=1}^{N_{func}} W_i} \tag{6.16}$$

where W_i is the weight assigned to a particular correspondence function. Depending on the relevance of each criterion, the user assigns weights and tolerances to each function ($W_{func} = 0$ means no evaluation at all). When two features are tested for correspondence, all correspondence functions with a positive weight are evaluated and the correspondence factor is returned. This factor has similar characteristics as equation 6.2, i.e. $-\infty \leq Corr(F_1, F_2) \leq 1.0$.

Thus, two features can be corresponded by assigning suitable weights and tolerances for each possible correspondence criterion. The resulting correspondence factor is a measure for the correspondence between the two features. A positive match between two features is found when the correspondence factor is positive.

6.3 Tracking using Prediction-Verification

Now that we have a measure to determine the correspondence between two feature, we can create a procedure to track features in time. Our tracking algorithm is based on a simple assumption: **features evolve predictably**. This implies that once a path of an object is found, we can make a prediction to the next frame and search for features in that frame that correspond to the prediction. If a candidate in the next frame corresponds to the prediction, the candidate is added to the path and a new prediction is made to the following frame. Thus, a continuous path can be found by prediction-verification until the path stops or until the last frame in the time series is reached.

The feature data representation allows the manipulation of features which makes it easy to calculate an extrapolation that can be used as a prediction. We calculate a prediction by linear extrapolation using the last two features in the path of an object:

$$F_{i+1} = F_i + \frac{(F_i - F_{i-1})}{(t_i - t_{i-1})} * (t_{i+1} - t_i) \tag{6.17}$$

where F_i is the feature in frame i and t_i is the time at frame i. When frames are equidistant in time, this equation becomes trivial. Other (higher order) prediction schemes are possible, but we believe that the gains do not counterbalance the higher computational costs. Furthermore, the linear extrapolation is in accordance with the heuristic of small velocity change, Section 6.1: the velocity of a point in the image changes little to the next time-step.

Figure 6.1 illustrates the process of prediction-verification. It shows six matched features that form the path of an object (dark objects), it shows the prediction at the end of the path (transparent object), and it shows a candidate feature (light object). Clearly, the candidate feature corresponds very well to the prediction and should be added to the path.

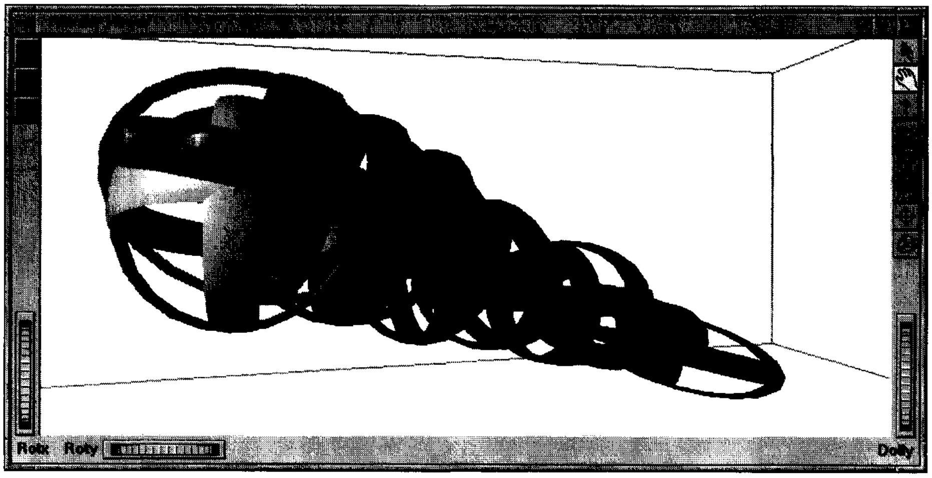


Figure 6.1: Visualization of a path (dark objects), its prediction (transparent object) and one of the candidates (light object).

6.3.1 Initialization

Yet, a prediction can only be made if a path exists, i.e. an initialization of new paths is required. The initialization is achieved by assuming a correspondence between two features in two successive frames. This assumption leads to a prediction that can be compared to all candidates in the third frame. If there is a candidate in the third frame that corresponds to the prediction, a new path is created and the path is continued into subsequent frames.

The algorithm of the initialization is shown in pseudo-code in Algorithm 1. It starts new paths from all features in the current frame (frame[i]) that have not been added to a path yet, i.e. the *unmatched features*. Basically, connections to all candidates in the next frame (frame[i+1]) are created, the predictions are calculated and corresponded to the candidates in the third frame (frame[i+2]). If there is a candidate that corresponds to the prediction, a new path is created and it is continued into subsequent frames.

Algorithm 1 Initialization, starting new paths from frame[i]

```
InitializePaths ( frame[i] )
{
    for_all ( unmatched features in frame[i] ) {
        for_all ( candidates in Frame[i+1] ) {
            assume connection between $feature_i$ and $candidate_{i+1}$
            calculate $prediction_{i+2}$
            for_all ( candidates in Frame[i+2 ]) {
                if ( Correspondence ( $prediction_{i+2}$, $candidate_{i+2}$) $> 0.0$ )
                    create new path
                    ContinuePath ( path, frame[i+3] )
                if ( path$\rightarrow$length $\geq$ MinPathLength )
                    add path to event graph
            }
        }
    }
}
```

There are two definitions for a candidate[2]: 1) all features in the next frame are candidates, and 2) only unmatched features in the next frame are candidates. The first option allows multiple solutions for one feature (it may be added to more than one path), and also requires an exhaustive search (all connections to the next frame are tested every time). The second choice limits the search because the number of candidates decreases when more paths are found, but it also removes a feature as possibility once a feature has been added to a path, i.e. the tracking solution depends on the order in which the features are tested. Section 6.3.2 describes a procedure that solves this problem by starting with strict tolerances and subsequent relaxation of tolerances in later passes.

A new path is created when a candidate in the third frame corresponds to the prediction. However, a match in the third frame could have been found coincidentally. Therefore, an additional test on the resulting path is performed

[2]Although in practice we always use unmatched features as candidates, we prefer to speak of candidates instead of unmatched features because the other definition is always an option to use.

to ensure authenticity: the path length must be larger than a given minimal path length (normally we take 4 or 5 frames). If the path length is larger than this minimal path length, we are confident that the path is genuine and we accept it by adding it to the event graph (discussed in Section 6.4).

The search for continuous paths may lead to multiple paths sharing the same feature in two ways: 1) multiple candidates correspond to the prediction and each leads to a new path, and 2) a candidate was already added to another path. This happens especially if the tolerances are relaxed. In case of multiple correspondences the path with the best *confidence index* gets the advantage. The confidence index of a complete path is calculated as follows:

$$\begin{aligned} Conf(\text{path}) &= 1 - e^{-\frac{C_{all}}{\tau}} \\ \text{with } C_{all} &= \sum_{i=1}^{N_c} Corr_i \end{aligned} \tag{6.18}$$

where N_c is the number of correspondences in the path, and τ is a growth factor (which is set equal to the minimal path length). The confidence index increases as the length of the path increases, which is convenient since, intuitively, our confidence in a path increases for longer paths. The confidence index is used when a choice has to be made between two paths sharing the same feature.

The worst case complexity of the initialization is of order $O(nm^2)$, with n the number of frames and m the number of features per frame which is usually much less than 100. It should be noted that the number of unmatched features from which a new path is started, decreases rapidly once a number of paths are found.

6.3.2 Multiple passes

Notice that the prediction-verification algorithm also works in the negative time-direction. Hence, we can search for continuous paths in two directions: in the forward and in the backward time-direction. This may result in new solutions, because forward and backward predictions are different. Therefore, it is useful to go back and forth through the frames in several passes. This process remains interactive because we use only the feature data. Interactive passes would not have been feasible if we had used the original grid data.

When a new pass is started with an existing tracking solution, the existing paths are tested first because they may be extended. There are two ways to extend a path. First, paths may be joined; if two paths start and end in successive frames, the two paths may be each other's continuation. Second, a path may be extended with candidates in the next frame. Before new paths are initialized, we have to check these extensions of paths first. Thus, an existing correspondence solution can be extended in multiple passes.

This also allows multiple passes with changed tracking parameters, for instance with different tolerances. Experience shows that good tracking results are obtained when tracking continuations is done in multiple (forward-and-backward) passes with linearly increasing tolerances. First, start with strict tolerances finding the obvious paths, and then relax the tolerances for each successive pass finding the more indistinct paths. With this scheme we can safely use the unmatched features as candidates, without being concerned about the order in which features are tested.

The user-interface (Section 6.6) offers the possibility to indicate the number of iterations in which the tolerances are increased. For each iteration the tracking process is repeated in forward and backward direction with the same tolerances until no new correspondences are found. Then, the next iteration is started with increased tolerances, until the tolerances reach the values that were provided by the user.

6.4 The Event Graph

[Samtaney *et al.*, 1994] proposed to use a relational graph, the Directed Acyclic Graph (DAG), to store the relations between corresponding features. We adopted the idea of using a relational graph, however we extended the DAG representation in such a fashion that it holds both the path-information and the frame-information, i.e. it provides possibilities to walk through the graph in both longitudinal and transverse direction (this will become clear later in this section). We call the extended DAG representation of the tracking results the correspondence graph or event graph[3].

The event graph representation is organized as follows. The building blocks of the graph are the nodes and edges, representing the features and their correspondences. Each node represents a feature in a certain frame and an edge indicates the correspondence between two features. Each edge has two pointers to the connected nodes. A node has a pointer to the `Feature` data of the feature in question, a feature number, a frame number, a list of incoming edges, and a list of outgoing edges. Thus, each feature in the data set is represented by a node, and each correspondence found by the tracking process is represented by an edge.

The path of an object is simply a collection of nodes and edges that are connected, and thus relate to the same object. The object id is set equal to the number of paths found so far, and every feature in the path gets this object id. The object path also holds information about the begin and end nodes of the path. Thus, one can run through the path frame by frame starting from a begin node and ending at an end node.

[3]So far, the name correspondence graph seems more appropriate, but the name event graph is used considering the event detection step, explained in Chapter 7.

Supplementary to the object path information, each frame is represented by a level in the graph. Each graph level holds a pointer to the `Frame` data; i.e. the collection of feature data in that frame. Also, for every feature in the frame it holds a list of nodes pointing to the feature[4], a list of unmatched features, a list of begin nodes, and a list of end nodes. Thus, we can easily run through the nodes in one frame and find all unmatched features, begin nodes, and end nodes in that frame.

The event graph representation holds both the object path and the graph level information. This provides us with full flexibility for handling the feature tracking results. We can ask a level to give all the nodes pointing to a certain feature and then find the paths that hold these nodes. Consequently, we can run through the graph in two directions: longitudinally by using the object path information, and transversely by using the graph level information.

6.5 Visualization of Time-Dependent Data

The last step in the visualization of time-dependent data (Figure 2.11) is the display itself. The tracking results should be displayed in a clear and meaningful fashion.

We use a combination of two viewers: the *Feature Viewer* and the *Graph Viewer*, which can be shown simultaneously. The feature viewer visualizes the features in 3D-space using iconic visualization techniques. The icons give an impression of the attribute values of a feature. The graph viewer visualizes the event graph in an abstract 2D representation. We believe that the combination of the two viewers provides an excellent way to visualize time-dependent data. It helps the user to understand relations between corresponding features, to view feature evolution, and to explore the time-dependent data.

6.5.1 Graph viewer

The graph viewer shows the event graph in an abstract 2D representation, showing the relationships between corresponding features. It is similar to the DAG-visualization in Figure 2.14, used in [Samtaney *et al.*, 1994], however with more utilities. In the graph viewer the features are represented by nodes and the correspondences between features by connecting lines (edges) between the nodes. The frames are plotted horizontally and the features are plotted vertically: (x, y) = (FrameNr, FeatureNr). The space between nodes is fixed, which results in a clear distinction between the individual features.

Figure 6.2 shows two different views of the same event graph: the graph drawn normally with the feature number on the y-axis (here the features are

[4]In case of multiple correspondences, the number of nodes pointing to a certain feature can be larger then one. It is essential to know when this happens.

sorted by size), and the graph drawn with minimal edge crossings which separates the paths more clearly. It is also possible to relate the vertical axis proportionally to a data value of one of the attributes of the feature (for instance size), which shows the attribute profile as a function of time.

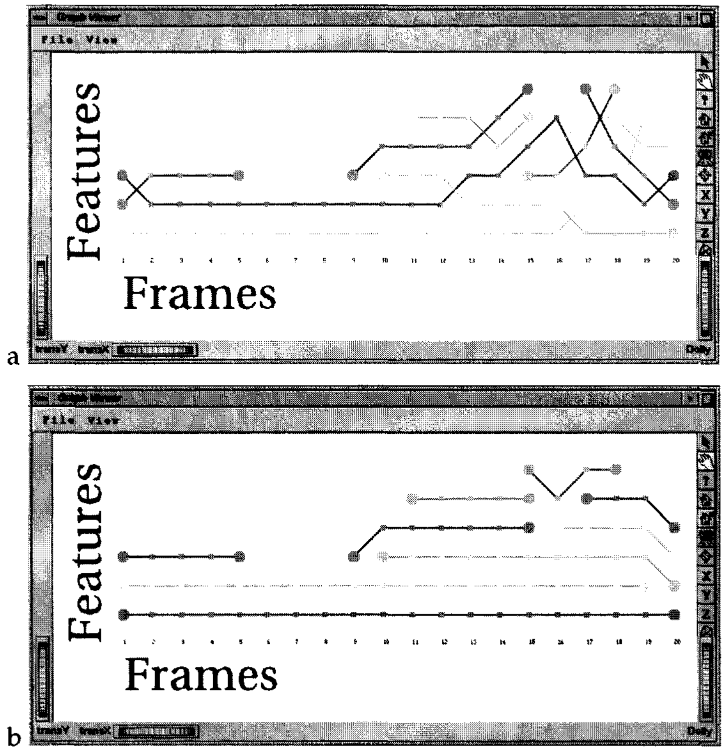


Figure 6.2: Two visualizations of the same event graph: a) the event graph drawn normally, and b) the event graph drawn with minimal edge crossings.

Colors are used to distinguish the different object paths in the graph. Unmatched features are shown in grey, while all feature nodes in the same path are shown with the same color.

6.5.2 Feature viewer

The feature viewer visualizes the features in 3D-space using iconic visualization techniques [van Walsum *et al.*, 1996]. The parameters of the icons are directly related to attribute values of a feature. Thus, the user can view the evolution of the attributes of the objects by animating the icons. A player was created to browse through the frames and to allow exploration of the time-dependent evolution of features in 3D. Figure 6.3 shows the player displaying one frame.

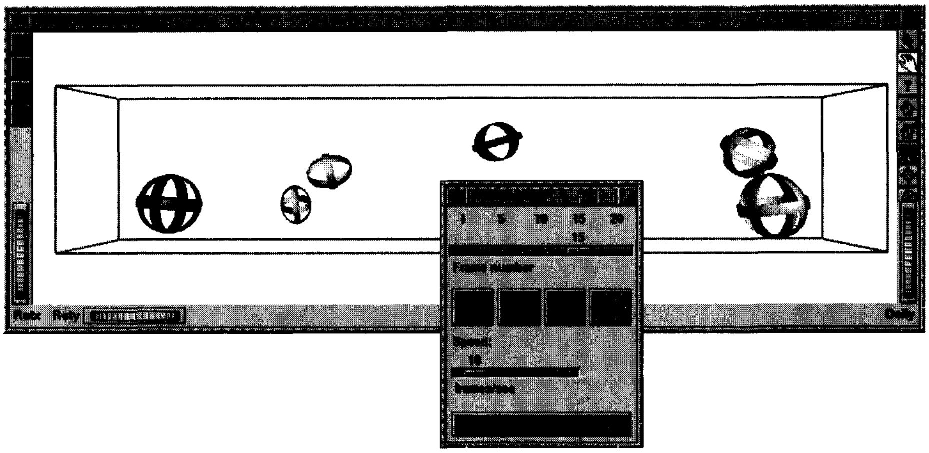


Figure 6.3: The frame player for browsing through the frames using the feature viewer.

6.5.3 Combination of two views

The linked combination of the graph viewer and the feature viewer is a powerful tool to explore time-dependent data. The same colors for features are used in both viewers which creates a direct link between nodes in the graph viewer and features in the feature viewer. Also, when a node is selected in the graph, the corresponding feature icon is shown in the feature viewer, and vice versa.

Figure 6.4 (shown in color on the backside of the cover) shows one step during the tracking process. The direct correlation between the two viewers provides a strong, interactive tool for visualizing and exploring time-dependent data. The viewers provide an overall visualization of time-dependent data, which facilitates the recognition of certain patterns in the evolution of the features. Also, they provide means for selective visualization of paths, nodes, and frames. The user can view additional information by selecting one particular path, frame, node, or edge.

6.6 Tracking User-Interface

Figure 6.5 shows the control panel that is presented to the user when the tracking process is started. With this control panel, we can set tracking parameters, start the tracking process, and view the intermediate and final results.

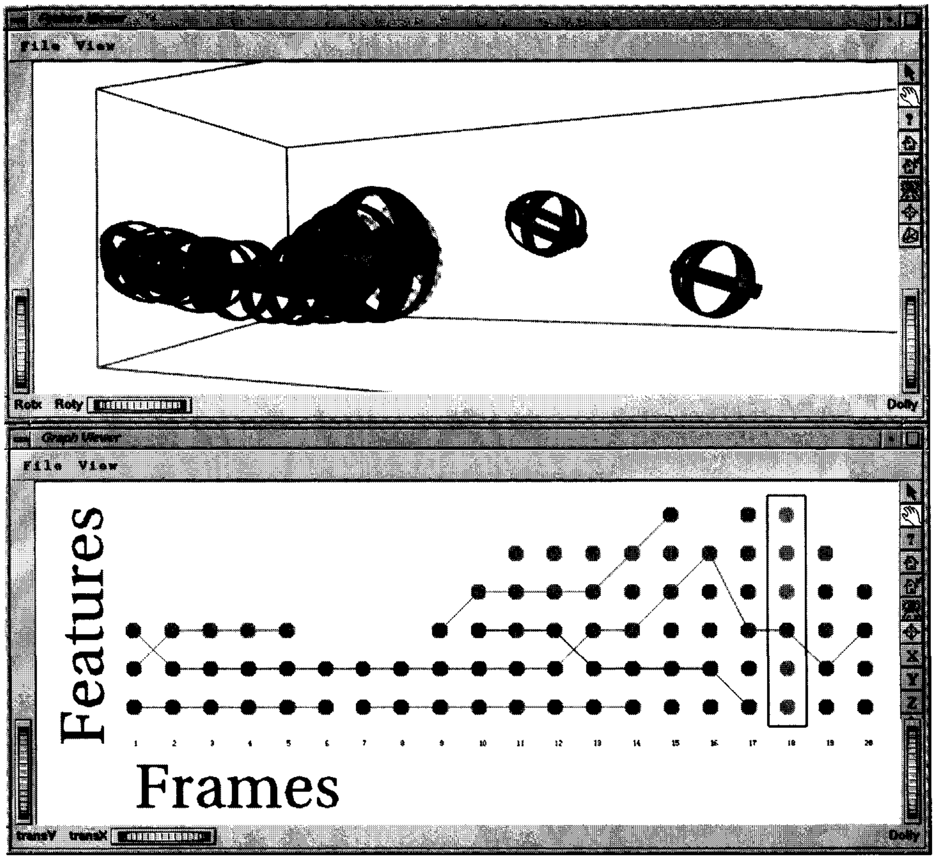
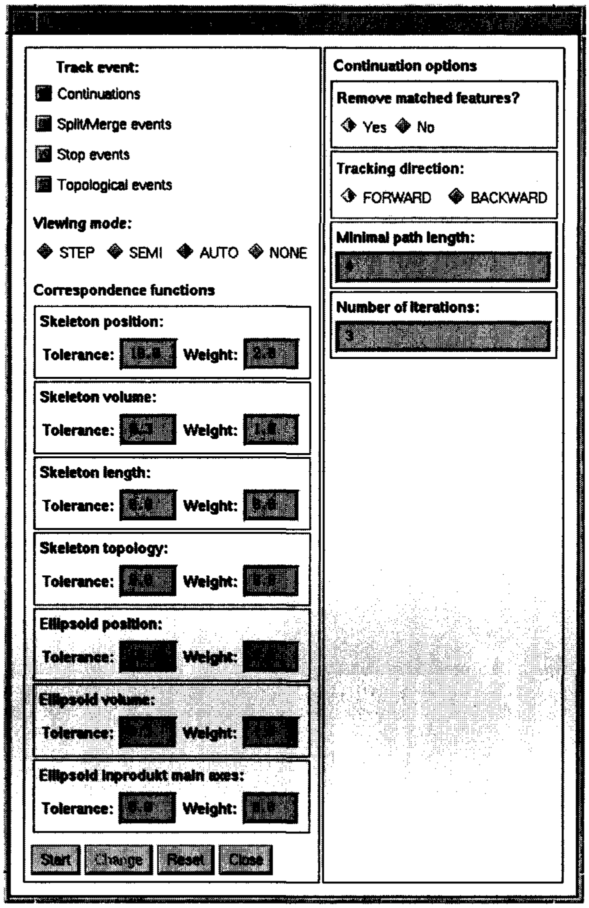


Figure 6.4: One step in the tracking of features showing the path (red objects), the prediction (transparent object), and a number of candidate features (blue objects), figure is shown in color on the backside of the cover.

The event options (top left) include the tracking of continuations, as well as the detection of other events. Below the event tracking specification, the viewing mode can be selected. The tracking process can be performed automatically, semi-automatically, and in stepwise control. In the stepwise and the semi-automatic mode, the user can examine correspondences in the two viewers and interactively steer the tracking by changing tracking parameters. Figure 6.4 shows one step in the tracking process.

Then, there are the controls for setting the correspondence functions. This part of the user-interface is built up depending on the attribute sets in the feature data. Here, the feature data consists of skeleton and ellipsoid data. The user can specify the tolerances and weights for every correspondence function, thus



Track event:

▨ Continuations

▨ Split/Merge events

▨ Stop events

▨ Topological events

Viewing mode:

◆ STEP ◆ SEMI ◆ AUTO ◆ NONE

Correspondence functions

Skeleton position:
Tolerance: 10.0 Weight: 2.0

Skeleton volume:
Tolerance: [illegible] Weight: 1.0

Skeleton length:
Tolerance: 0.0 Weight: 0.0

Skeleton topology:
Tolerance: 0.0 Weight: 0.0

Ellipsoid position:
Tolerance: [illegible] Weight: [illegible]

Ellipsoid volume:
Tolerance: [illegible] Weight: [illegible]

Ellipsoid inprodukt main axes:
Tolerance: 0.0 Weight: 0.0

Continuation options

Remove matched features?
◆ Yes ◆ No

Tracking direction:
◆ FORWARD ◆ BACKWARD

Minimal path length:
4

Number of iterations:
3

Figure 6.5: The tracking control panel, where tolerances, weights, and tracking parameters are defined.

deciding which criteria are relevant. It is possible to track continuations based on the skeleton data first, and then additionally track continuations based on the ellipsoid data.

On the right side of the control panel the parameters of the tracking process can be controlled. We can choose the definition for a candidate, the tracking direction, the minimal path length, and the number of iterations. Also, here the buttons are shown that control the tracking process when the viewing mode is set to stepwise, or semi-automatic.

The buttons (on the bottom-left) are used to start the tracking process, trigger the change of parameters, reset the event graph, and close the tracking control panel.

6.7 Comparison to other Tracking Methods

In Section 2.6.2 three methods for feature tracking were discussed in more detail; attribute tracking by [Samtaney *et al.*, 1994], volume tracking by [Silver & Wang, 1997], and greedy exchange tracking by [Sethi *et al.*, 1994]. Here, we will compare these methods to our tracking approach.

6.7.1 Attribute tracking

Attribute tracking by [Samtaney *et al.*, 1994] uses only the attributes of features to find their evolution. The correspondence criteria are similar to the criteria described above. However, the attribute set is simply a fixed set of attributes. If an attribute is not calculated because it is not relevant for this specific feature, it is simply not given any value, but it does exist in the data structure. Our approach with the feature data representation is much more generic.

The tracking method is entirely performed on a two-frame basis. The features in frame i are corresponded to the closest (in position) features in frame $i+1$. The underlying assumption is that the time between successive frames is small (the sampling frequency is high enough to capture object motion and evolution) and thus the object positions do not change much.

This two-frame approach has two disadvantages. First, it lacks sense of continuity of motion; information about previous motion of the objects is not used in tracking. This may lead to incoherent paths, for instance when two feature paths cross each other, see Figure 6.6. Second, the algorithm cannot track fast moving objects because their positions in two successive frames are not 'close' to each other. Also, this eliminates the possibility of adaptive temporal sampling (the removal of frames as discussed in Section 6.8), because the assumption of small time-intervals is not satisfied.

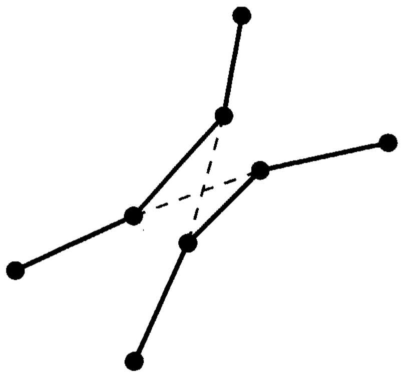


Figure 6.6: The tracking of two paths that cross each other may lead to incoherent paths. The coherent paths are indicated by the dashed line.

6.7.2 Volume tracking

Volume tracking by [Silver & Wang, 1997; Silver & Wang, 1998; Silver & Wang, 1999] uses a representation for the segmented volume of the features (an octree representation in the earlier work, and a linked-list representation in later work) in order to determine overlap (in space) between features in successive frames. The overlap test results in a number of features in the next frame that are tested for correspondence. Correspondence is established by testing the attributes of the features with the same criteria as in [Samtaney *et al.*, 1994]. Thus, the algorithms are similar, but overlap replaces the proximity criterion.

The advantage of the overlap test is that corresponding features most likely overlap, while their position may not be close to each other. Chapter 5 showed that average position may be a misleading attribute for features with a certain shape. The average position of the turbulent vortex structure (Section 8.3) could even be outside the object's geometric shape. In this application, spatial overlap provides a much better indication for correspondence than proximity in position, resulting in better tracking solutions.

The disadvantage of the overlap test is that it is memory consuming. For each feature a binary volume representation needs to be saved. Although the volume representation consumes less memory than the binary grid representation, it still needs far more memory than the attribute representations. On the other hand, it provides an accurate description of the feature's spatial extent and shape (even better than the skeleton graph representation).

Besides the memory consumption, it suffers from the same disadvantages as the attribute tracking. It is based on a two frames correspondence and therefore lacks sense of continuity of motion. And fast moving small features may be missed because they do not overlap. Therefore, the assumption of small time-intervals is still needed. The advantage of the method is that it is robust and relatively fast compared to direct visualization of the raw data.

We obtained tracking results for a CFD application with turbulent vortex structures using the volume tracking[5]. The results were transformed to our feature data and event graph representations, so that we could compare the results of the two algorithms. This comparison is described in Section 8.3.2. The most important conclusion is that the two graphs were 91% in agreement with each other. This shows that tracking results are comparable even with two different tracking methods.

[5]Data courtesy to D. Silver and X. Wang of Rutgers University.

6.7.3 Greedy exchange tracking

The greedy exchange method by [Sethi *et al.*, 1994] can be used to track tokens in a k-dimensional property space. It employs directional coherence and speed coherence, resulting in smooth trajectories (paths) corresponding to the preferences of the human visual system for continuity of motion. However, it uses the attributes as individual parameters in a k-dimensional property space where attributes are not related to each other and have no additional significance.

As the name greedy exchange suggests, the method requires a large number of operations because it uses an optimization process. The algorithm basically searches for the maximum motion smoothness over all trajectories. The algorithm starts by creating initial trajectories by connecting the closest tokens in property space. Then tokens are exchanged iteratively in a systematic fashion until the maximum smoothness is found. Tokens are only exchanged if they lie within a certain maximum distance d_{max} in property space. This means not every possible trajectory is tested, but still the number of tested possibilities is large.

We implemented the greedy exchange tracking method in such a way that it could work with our feature data representation. The position and volume information in the ellipsoid attribute set are mapped on the axes of a 4-dimensional property space. Thus, we were able to track features with the greedy exchange method and compare the results to our prediction-verification method.

Comparison experiments, which are described by [Vrolijk, 2000], were done on a number of applications. One of the applications is the flow past a tapered cylinder discussed in Section 8.4. Tracking was performed on the features by both methods in order to find the tracking parameters that lead to the best correspondence between the two methods. With the best choice of parameters we could find a graph correspondence of 86%. This is a little lower than the results from the volume tracking, but it is still in reasonable agreement with each other.

The performance of both methods is tested by measuring the time needed to complete the track as a function of the number of frames. Tracking is performed on a selection of frames from the complete time series on an SGI Octane with one 225 MHz R10000 processor and 256 Mb of memory. Figure 6.7 shows the results of the tests.

The figure shows that the greedy exchange method from [Sethi *et al.*, 1994] needs more computation time than our prediction-verification method [Reinders *et al.*, 1999a]. The complexity of the greedy exchange has a quadratic behavior which is caused by the number of trajectories that increases with the number of frames. Clearly, the times of the prediction-verification technique show a linear increase with the number of frames. This is a huge advantage of our algorithm compared to the greedy exchange method.

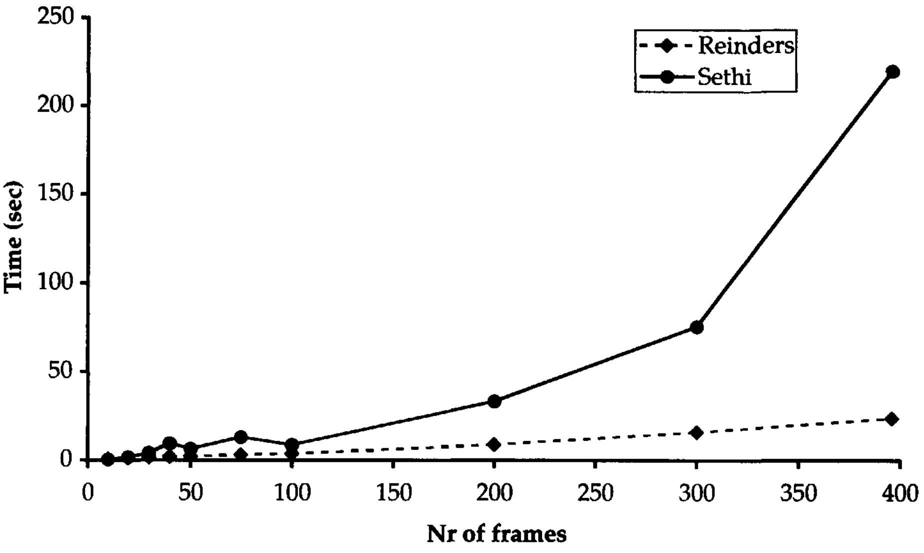


Figure 6.7: The time needed for the algorithms to complete the track as a function of the number of frames.

6.8 Future Prospects

Although, the current (linear) prediction scheme is in accordance with the heuristic of small velocity change, it is easy to adapt our tracking algorithm to other prediction schemes. For instance, a prediction scheme based on a second-order extrapolation, or the incorporation of a-priori knowledge, e.g. the mainstream of a flow. This may lead to more accurate predictions depending on the continuity of motion. A disadvantage is the increase of complexity with the order of prediction.

The sampling of the frames does not necessarily have to be equidistant in time; it is possible to apply temporal refinement. The tracking process is not influenced by this because the prediction-verification scheme is independent of the time between frames; see equation 6.17. The time between frames may be varied according to the variance of the data in time. Turbulent time-periods may require a dense sampling in time, while periods with less turbulence can be simulated with a sparse sampling, i.e. similar to local (spatial) refinement of the grid.

Temporal refinement may increase the efficiency of the analysis of evolutionary phenomena. Currently, time-dependent numerical simulations often calculate far too many time steps in order to obtain a certain accuracy. Often, 90% of the data sets is thrown away because they consume too much disk space and they are not necessary for the analysis of evolutionary phenomena.

With feature tracking, it is possible to automatically recognize the episodes in the time series where interesting phenomena happen. It should be possible to restart the simulation from a key-frame at the beginning of such a period and to calculate new time steps with a denser sampling in time. Thus, we can focus on a small time-interval and investigate the evolving phenomena in more detail. Perhaps it is even possible to focus on the evolution of one feature in a small period in time in a limited range in space, i.e. to refine both in space and in time.

The tracking results can also be used to detect 'weak' features in the data. Weak features are features that are weak with respect to the selection criteria that the data should satisfy. The data values at the grid nodes lie close to the selection threshold and it is doubtful if such a feature is detected or not. In order to improve the detection, we can use the prediction to the next frame. The prediction tells us where the weak feature can be found and assists us in finding the right selection criteria for detecting this feature. In this fashion, there is a direct coupling between the tracking process and the feature extraction step.

The visualization in two views is usable during the tracking process; new correspondences can be found by visual inspection (user-guided tracking). The linked combination of the graph viewer and the feature viewer is a powerful tool to explore correspondences. It should be possible to tentatively create a connection between two nodes in the event graph *graph editing*, view the two features in the feature viewer, and verify the correspondence visually and quantitatively.

Chapter 7

Event Detection

Feature tracking solved the one-to-one correspondence problem between features in subsequent frames, resulting in continuous paths of objects. The features in the path hold the data describing the characteristics of the object in each frame. The evolution of an object can be investigated by studying the feature attributes of the features in the path. For each object a description can be made of its lifespan, describing its origin, motion, growth, decay, and so forth.

The investigation of the evolution of an object automatically focuses on significant evolving phenomena: *events*. An event can be defined as any development in the evolution of a feature that is significant. Examples of events are: unusual changes in the attributes, particular stages in the life cycle of a feature, specific interactions with other features, or periodic patterns in the evolution. The events can be interpreted as temporal features in the evolution of features. It raises the level of abstraction to yet another higher level; from data describing the characteristics of features to data describing the evolution of features.

Event detection tries to detect these events in an algorithmic, automated fashion. The different types of events depend on the physical phenomena underlying the dynamics of the feature. For instance, some features may not be able to merge, but collide when they meet, or the boundary of the system may be closed so that entry and exit events (Section 7.3.2) are impossible. Each different type of event is translated into criteria that are tested by an algorithm. The criteria should be based on the underlying physics of that event.

This chapter discusses event detection based on the feature data representation and the feature tracking procedure as discussed in Chapter 6. The chapter describes the different types of events that we can detect, the underlying criteria, the detection algorithm, and the visualization of these events.

7.1 Different Types of Events

There exist many different types of events depending on the application, the type of features, and on the user's interest. [Samtaney *et al.*, 1994] introduced a number of evolutionary events: continuation, creation, dissipation, bifurcation, and amalgamation. Figure 7.1 shows a schematic representation of these events. With some imagination, we can think of many other types of events (a number of them is discussed in Section 7.8). It is impossible to create a generic method to detect any type of event because the criteria differ too much and the detection process is different. Therefore, we provide a number of events that can be detected. The user can select one or more depending on the application and his interest.

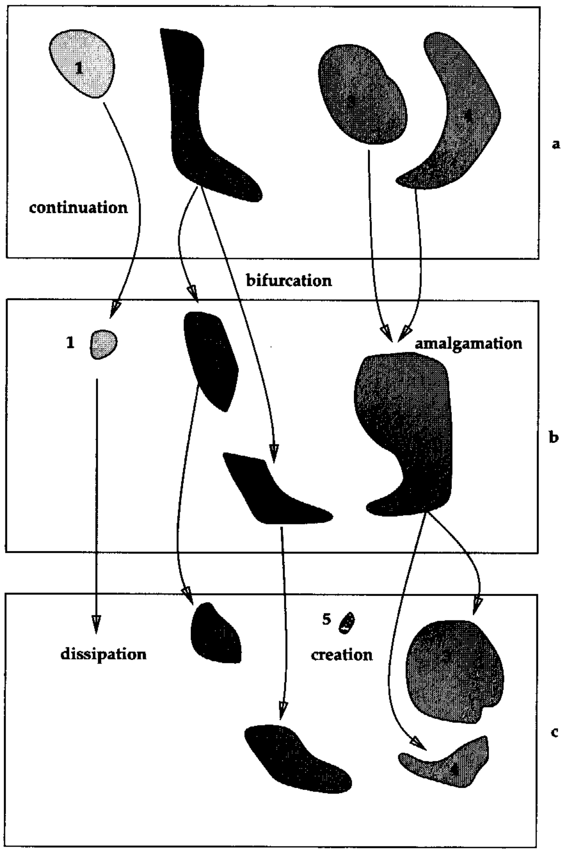


Figure 7.1: Tracking interactions: continuation, creation, dissipation, bifurcation, and amalgamation, [Samtaney *et al.*, 1994].

Currently, our event detection system recognizes the following events[1]:

- **Continuation.** One-to-one correspondence found between two features in subsequent frames. The feature continues without dissipating or interacting with other features.

[1]Compare the terminology to [Samtaney *et al.*, 1994]: Continuation, Creation = Birth, Dissipation = Death, Bifurcation = Split, and Amalgamation = Merge.

- **Birth/Death.** A feature at the start or at the end of a path decreases in size and dissipates. The related event is called birth or death, depending on the direction of time.
- **Entry/Exit.** A feature at the start or the end of a path moves through an open boundary of the system (see Section 7.3) and disappears. The related event is called entry or exit, depending on the direction of time.
- **Split/Merge.** Two or more features merge and continue as one, or one feature splits and continues as two or more. The related event is called merge or split, depending on the direction of time.
- **Topo-Loop.** The topology of a feature changes: a loop appears or disappears. The number of loops in the topological graph changes.
- **Topo-Junction.** The topology of a feature changes: a junction appears or disappears. The number of edges in the topological graph changes.
- **Unresolved.** If none of the above events can give an explanation, the feature event is unresolved.

Note that most events come in pairs because they are each other's reverse in time. This means that a split can be found by testing the criteria for a merge in the reverse time-direction.

Our event detection system detects the events described above by a prediction-verification scheme, similar to the way the features were tracked in Chapter 6. Thus, again the assumption is that features evolve predictably. Each event that can be detected has its own criteria for prediction-verification.

The criteria for a specific event are tested by certain *event functions*. Each attribute set (see Section 3.2.2) in the feature data representation is associated with a number of functions for each specific type of event. If there are no functions associated to the attribute set for a certain event, then that event cannot be detected as far as this attribute set is concerned.

Below we will discuss each different type of event. Per event we explain the underling criteria, the event functions, and the detection algorithm.

7.2 Continuations

Continuations may be considered as non-events, they are the result of the feature tracking process described in Chapter 6. However, this step is necessary before the real event detection can take place. Continuation detection may have to be repeated after the event detection results in new correspondences. For instance, when two paths merge into an unmatched feature, this leads to a new ending of a path, which may be continued into the next frame. The result of the tracking algorithm is a number of continuous paths that can be used for the detection of real events.

The event function that is needed to detect continuations is the correspondence function C_{func} described by equation 6.2. Each attribute set is associated to a number of these functions as was described in Section 6.2.1. The correspondence functions have the following properties: $C_{func}(F_1, F_2) = 1$ when the attributes in the two features are identical, $C_{func}(F_1, F_2) = 0$ when the deviation is on the limit of the tolerance, and $C_{func}(F_1, F_2) \leq 0$ when the deviation exceeds the tolerance.

Figure 6.2a shows the result after tracking the continuations. It shows only continuous paths, the objects continue without interaction with others, or other events.

7.3 Terminal Events

After continuous paths have been found, the end nodes of the paths can be explained by terminal events, i.e. birth/death or entry/exit. Each end node of a path in both directions of time can be tested to satisfy the criteria for a specific terminal event. These criteria are as follows.

7.3.1 Birth/Death event

The physical rules for a birth or a death can be defined as follows: a feature at the start or at the end of a path decreases in size and disappears. The related event is a birth or death, depending on the time-direction. Thus, for a death two requirements must be satisfied on the end feature F_{end} (frame i) and the prediction feature F_{pred} (frame $i+1$):

1. The size at the end of a path must decrease. This is a test on the difference in size (volume or mass) between the prediction V_{pred} and the end node V_{end}:
$$V_{pred} < V_{end} \tag{7.1}$$
2. The size of the prediction must be small. If size of the prediction is negative, the death event was expected and the death event function D_{func} should return a factor of 1.0. Otherwise, the volume of the prediction is tested relative to the volume of the end node:
$$D_{func}\left(V_{pred}, V_{end}\right) = 1 - \frac{V_{pred}/V_{end}}{T_{vol}} \tag{7.2}$$
The same tolerance T_{vol} can be used as in the volume tests for the continuations.

Thus, the death functions have as input the prediction and the end feature in the path and return the following values:

$$D_{func}\left(V_{pred}, V_{end}\right) = \begin{cases} 1 & \text{when } V_{pred} \leq 0 \\ 1 - \frac{V_{pred}/V_{end}}{T_{vol}} & \text{when } V_{pred} > 0 \end{cases} \tag{7.3}$$

The values of the death functions are combined into one *death factor* C_{death} similar to the correspondence factor in equation 6.16, i.e. the functions are weighted. A death is detected when the death factor is positive: $C_{death} \geq 0.0$.

A birth is found as the reverse detection of a death: the death criteria are simply tested in the reverse time-direction. An example is shown in Figure 7.2. It shows the path as opaque objects and the prediction at the end of the path is shown transparently. The criteria for a birth are satisfied because the path stops (in reverse direction) and 1) the growth in size is negative, and 2) the prediction is small.

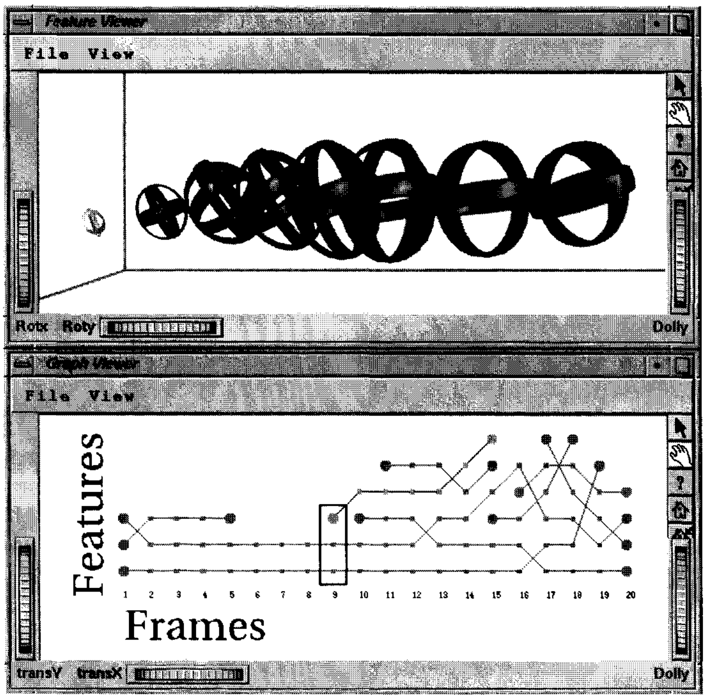


Figure 7.2: The detection of a birth event is performed in the reverse time-direction. The path is shown as opaque objects, and the prediction at the end of the path is shown transparently.

7.3.2 Entry/Exit event

The physical rules for an entry or an exit can be defined as follows: a feature at the start or at the end of a path moves through an open boundary of the system and disappears. The related event is called entry or exit, depending on the time-direction. An 'open boundary' is a system boundary where features may move through. Some domains have open boundaries, for instance a flow through a pipe has an inlet and an outlet. The inlet and the outlet are open boundaries, but the wall of the pipe in between is a solid wall, and therefore is a closed boundary.

Based on these physical rules we can formulate two requirements for an exit that must be satisfied. It is sufficient if these requirements satisfy for just one open part of the system boundary. The requirements involve the end feature F_{end}, the prediction feature F_{pred}, and the open parts of the system boundary S_{bnds}:

1. The object must move towards an open boundary. This can be tested by the position of the end node $\mathbf{P}_{end}$, the position of the prediction $\mathbf{P}_{pred}$, the boundary position $\mathbf{P}_{bounds}$, and boundary normal $\mathbf{N}_{bounds}$ that is directed outward of the system:

$$\begin{array}{lcl} \|\mathbf{P}_{pred} - \mathbf{P}_{bounds}\| & \leq & \|\mathbf{P}_{end} - \mathbf{P}_{bounds}\| \\ \wedge & & \\ (\mathbf{P}_{pred} - \mathbf{P}_{end}) \cdot \mathbf{N}_{bounds} & \geq & 0.0 \end{array} \tag{7.4}$$

   where $\|\mathbf{P}_{pred} - \mathbf{P}_{bounds}\| = 0$ when the position of the prediction is outside the system boundary.

2. The position of the prediction must be beyond or near the open boundary. If the position is beyond the boundary, the exit event is expected and the exit function E_{func} should return a factor of 1.0. Otherwise the position of the prediction is tested against the position of the boundary:

$$E_{func}\left(\mathbf{P}_{pred}, \mathbf{P}_{bounds}\right) = 1 - \frac{\|\mathbf{P}_{pred} - \mathbf{P}_{bounds}\|}{T_{pos}} \tag{7.5}$$

   Again, a similar tolerance T_{pos} can be used as in the position test for continuations. This tolerance can be imagined as a zone with a thickness of T_{pos} along the open boundaries were entries and exits are accepted.

The input of the exit functions are the prediction, the end feature in the path, and the system boundary. The functions should return, given the requirements above, a factor with the following properties:

$$E_{func}\left(\mathbf{P}_{pred}, \mathbf{P}_{bounds}\right) = \begin{cases} 1 & \text{when } \mathbf{P}_{pred} \text{ is beyond the boundary} \\ 1 - \frac{\|\mathbf{P}_{pred}-\mathbf{P}_{bounds}\|}{T_{pos}} & \text{otherwise} \end{cases} \tag{7.6}$$

The values of the exit functions are combined into one *exit factor* C_{exit} by weighting the results of the exit functions. An exit is detected when the exit factor is positive: $C_{exit} \geq 0.0$.

An entry is found as the reverse detection of an exit: the exit criteria are simply tested in the reverse time-direction. An example is shown in Figure 7.3. It shows the path as opaque objects, the prediction at the end of the path is shown transparently, and the system boundaries are also shown. The criteria for an entry are satisfied because the path stops and 1) the velocity is directed outward, and 2) the prediction lies clearly outside the system boundary.

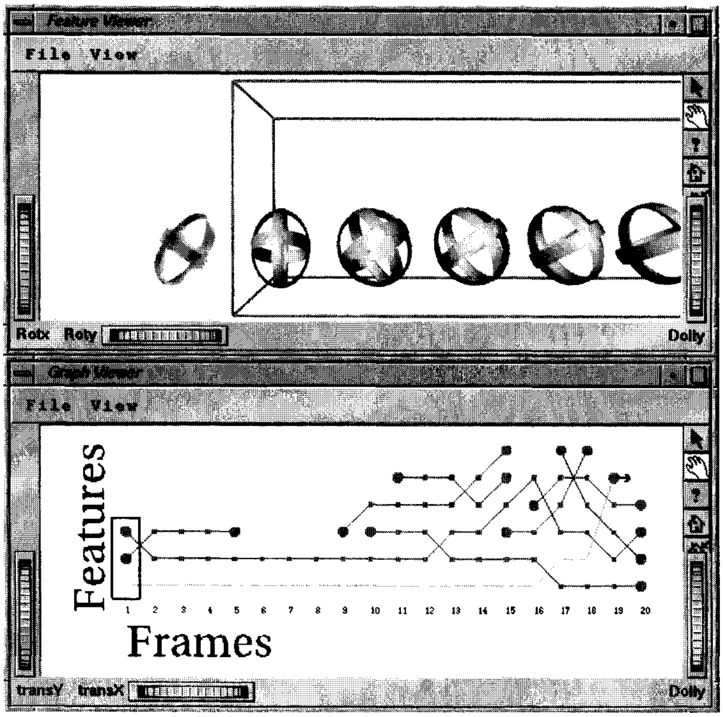


Figure 7.3: The detection of an entry event is performed in the reverse time-direction. The path is shown as opaque objects, and the prediction at the end of the path is shown transparently.

Since the death-test is essentially a volume test and the exit-test is essentially a position test, both tests can be used simultaneously. The test with the highest result is taken. If both tests give equal values, the preference of the user determines the terminal event. It is also possible to test each terminal event separately. Figure 7.4 shows the result after the detection of terminal events is executed. It shows that some of the end nodes of paths are explained by a birth/death or entry/exit events. The icons used are shown more clearly in Section 7.7.

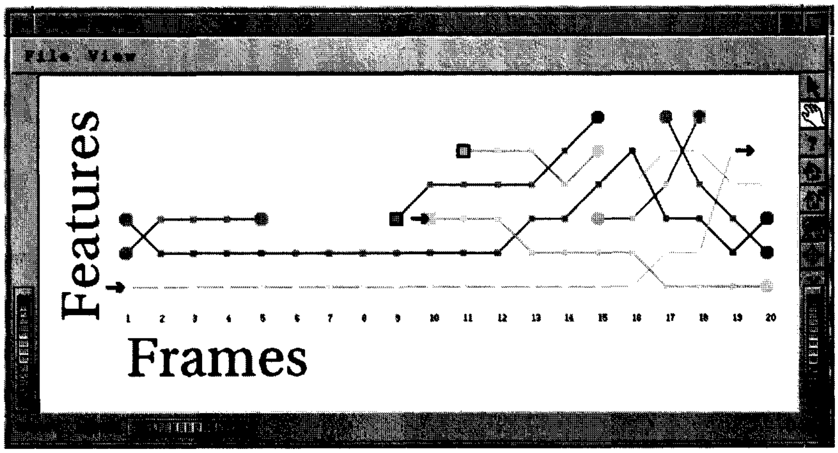


Figure 7.4: The resulting event graph after the detection of terminal events: a number of the path endings is explained.

7.4 Interaction Events

The evolution of features may be influenced by the presence of other features. Features may attract or repel each other, they may fluidly merge with each other, or they may bounce. The way features interact depends on the application and the type of features involved. Sometimes, the question is how the features interact, or whether a particular interaction occurs. Most of the time the user probably has some idea what will happen when two features meet.

The interaction of features depends very much on the underlying physical model. For instance, when features are diffuse (such as clouds) they will probably merge, and when they are solid objects they may bounce. Depending on these scenarios we can make a model to calculate the predictions for the next time-step. When features are expected to merge, the model should tell us how the attributes are combined. The prediction can then be tested with candidates in the next frame using normal correspondence functions. Here, we will discuss only the split/merge interaction of features.

7.4.1 Split/Merge event

The merge event can be defined as follows: two or more features approach each other, they merge and continue as one. The split event is the reverse of a merge: one feature splits and continues as two or more. When two objects merge, the result is an object with mass (or volume) equal to the combined masses (or volumes) and a position equal to the weighted positions weighted by the mass (or volume). This model for feature interaction is the underlying model for a split/merge event. Other behavior could also be modeled, leading to different types of interaction event.

Basically, the merge event is detected as follows: the predictions of two or more merging features are 'merged' into a single prediction. This prediction is tested with a candidate in the next frame with the normal correspondence functions. The merged prediction is calculated according to the model described above, it will be explained in more detail later in this section.

Possible scenarios for a merge

Starting from the result of the feature tracking process we can recognize three possible scenarios of how a merge may occur. Figure 7.5 illustrates these three scenarios:

1. Two or more paths end in frame i and merge into one unmatched feature in frame $i+1$.

2. Two or more paths end in frame i and merge into one start of a new path in frame $i+1$.

3. One path end in frame i and merges into one continuous path.

These three scenarios should be tested in this order, because the first two scenarios solve more unresolved connections than the last scenario. The first two are only possible if more than one path ends in the current frame, while the third scenario needs only one path ending in that frame.

If more than two paths end in frame i, all possible combinations of end nodes are tested to the unmatched features (scenario 1), or the start nodes (scenario 2) in the next frame. We accept only the combination with the best (positive) correspondence. In case of scenario 3, the end node of the path is tested one-by-one to every continuous path. The merge with the path that results in the best correspondence is accepted only if the local correspondence improves.

The number of tests for merge events can become substantial. In order to limit this number we only test features for a merge if they are close to each other. It is useless to test a merge between features that lie far away from each other, this can only lead to false positives.

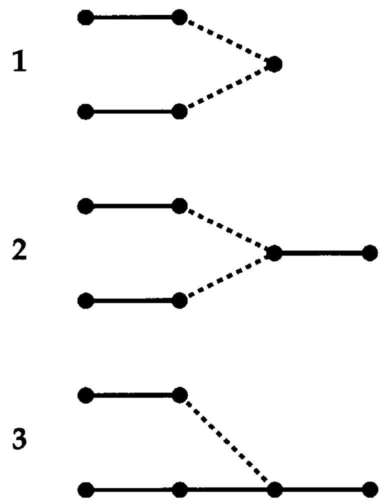


Figure 7.5: Schematic representation of the three possible merge scenarios. The dotted lines indicate the possible connections that need to be tested.

Neighborhood criteria

We use neighborhood criteria to decide which features are candidates for a merge. Only combinations are made of end nodes that have a prediction in the neighborhood of the candidate (scenarios 1 and 2). Also, an end node can only merge into continuous paths (scenario 3) that are in the neighborhood of the prediction of the end node in question. The neighborhood criteria test if two features are close to each other in position.

The neighborhood criteria are tested by neighborhood functions that are associated to each attribute set. The neighborhood functions are similar to the event functions, only they do not detect an event but only establish closeness. If an attribute set contains position information, it can be used to establish neighborhood. Table 7.1 shows the neighborhood functions associated with the ellipsoid and skeleton attribute sets.

For the ellipsoid the neighborhood criterion tests the distance between the two center positions of the ellipsoids. It is equal to the position test of the correspondence criteria of the ellipsoid. There are three different neighborhood criteria associated to the skeleton attribute set. The first tests the distance in the position of the center of gravity, the second determines the minimal node-to-edge distance $\min(D_{n-e})$ between the two skeleton graphs, and the third determines the minimal edge-to-edge distance $\min(D_{e-e})$.

For each of the neighborhood functions shown in Table 7.1 we have to assign weights and tolerances. Then, the neighborhood factor is calculated similar to the correspondence factor, by weighting the contributions of each criterion. The

Ellipsoid	$$\|\|\mathbf{P}_2 - \mathbf{P}_1\|\| \leq T_{pos} \quad (7.7)$$
Skeleton	$$\|\|\mathbf{P}_2 - \mathbf{P}_1\|\| \leq T_{pos} \quad (7.8)$$ $$\min(D_{n-e}) \leq T_{pos} \quad (7.9)$$ $$\min(D_{e-e}) \leq T_{pos} \quad (7.10)$$

Table 7.1: The neighborhood criteria associated with the ellipsoid and skeleton attribute sets, min (D_{n-e}) is the minimal node-to-edge distance, and min (D_{e-e}) is the minimal edge-to-edge distance.

tolerance values can be set similar to the values given for the correspondence factor.

Figure 7.6 shows two time steps with a merge between two features. Figure 7.6a and b show the skeletons, and c and d show the ellipsoids. The distance between the positions of the ellipsoids is large, while the distance between the skeleton graphs is small. The skeleton neighborhood criteria in equations 7.9 and 7.10 are likely to satisfy easier (with a smaller tolerance) than the ellipsoid neighborhood criterion. This is also shown in Table 7.2 which shows the minimal tolerance T_{pos} that finds the merge for each of the four neighborhood criteria[2].

Neighborhood criterion	Minimal tolerance T_{pos}
Ellipsoid position	40.8
Skeleton COG	36.8
Skeleton node-to-edge	0.6
Skeleton edge-to-edge	0.4

Table 7.2: The minimal tolerance for which the merge in Figure 7.6 was found for each neighborhood criterion.

The minimal tolerance for the neighborhood criteria that use the graph information are clearly much smaller than for the criteria that use the distance in position. Thus, with the skeleton attribute set we find the merge events with much smaller tolerances, reducing the chances of false positives.

[2]N.B. The position tolerance T_{pos} is an absolute tolerance, it is not a relative measure. Hence, they can be compared for each criterion.

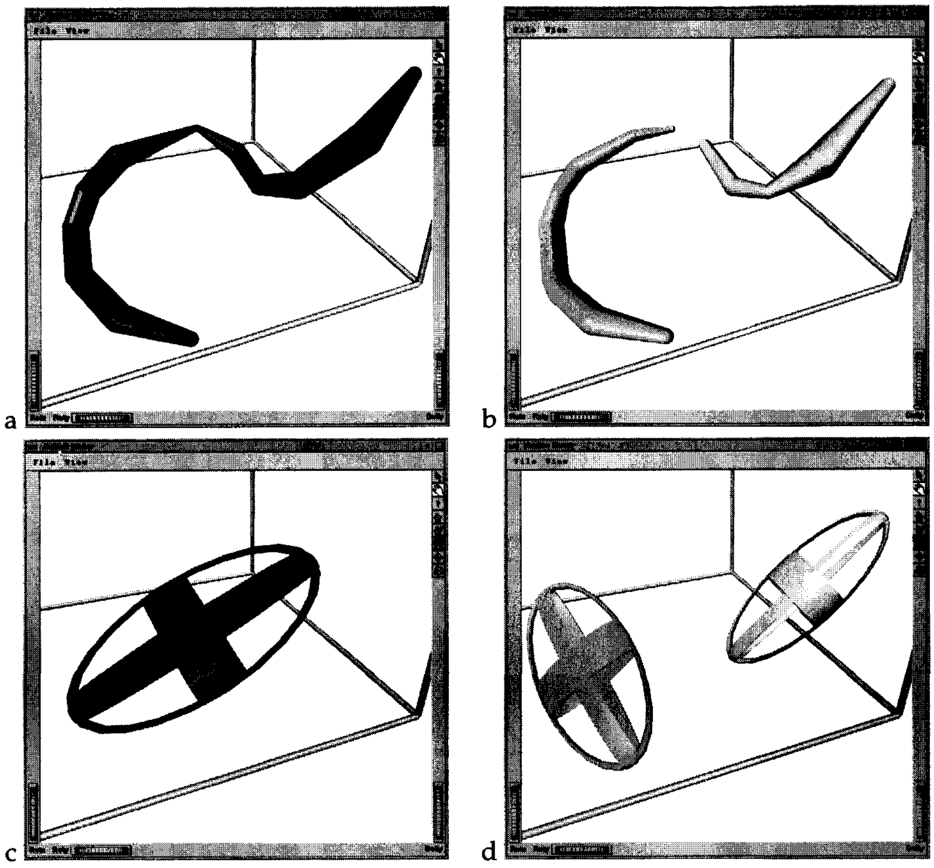


Figure 7.6: For line-type features, the skeleton attribute set provides better neighborhood criteria than the ellipsoid attribute set. This is important for the detection of merge events.

Calculating a merged prediction

Yet, we have to merge a number of predictions into one prediction that is tested with the candidate. The merged prediction is calculated based on the rules for a split/merge mentioned above. Volume and mass attributes are added and position is weighted by volume (or mass, if volume is not available). This is in accordance with physical laws such as conservation of mass. Table 7.3 shows the merge functions of the ellipsoid and skeleton attribute sets. The merge functions of each possible attribute set are defined in a similar fashion.

Attribute Set	Attribute	Merge functions
Ellipsoid	Position	$\mathbf{P}_{merge} = \frac{\sum_i \mathbf{P}_i V_i}{\sum_i V_i}$ (7.11)
	Axes	$\mathbf{R}_{merge} = \frac{\sqrt[3]{\sum_i V_i}}{4/3\pi}\mathbf{1}$ (7.12)
	Orientation	$\mathbf{A}_{merge} = \mathbf{0}$ (7.13)
Skeleton	Volume	$V_{merge} = \sum_i V_i$ (7.14)
	Position COG	$\mathbf{P}_{merge} = \frac{\sum_i \mathbf{P}_i V_i}{\sum_i V_i}$ (7.15)
	Length	$L_{merge} = \sum_i L_i$ (7.16)
	Graph	$G_{merge} = \sum_i G_i = \sum_i Nodes_i + \sum_i Edges_i$ (7.17)

Table 7.3: The merge functions for the ellipsoid and skeleton attribute sets. These functions are used to calculate a merged prediction.

Algorithm for merge detection

With the neighborhood criteria and the calculation of a merged feature, we can test the three scenarios for a merge. The algorithm tests for all frames i the three scenarios as follows:

1. For each unmatched feature in frame $i+1$ we find the ends with a prediction in the neighborhood of the unmatched feature. All possible combinations of these ends are tested for a merge.

2. For each unexplained start of a path in frame $i+1$ we find the remaining unexplained ends with a prediction in the neighborhood of the start. All possible combinations of these ends are tested for a merge.

3. The remaining unexplained ends in frame *i* are tested to merge with a continuous path that lies in the neighborhood of the prediction of the end.

Thus, for every frame in the time series the unexplained ends may be explained by a merge event.

Figure 7.7 shows one step in the detection of a merge with scenario 2. Two ending paths merge into the start of a new path. The figure shows the two ending paths and the start of the new path as opaque objects, and the merged prediction is shown transparently. Clearly, the start of the new path corresponds very well to the merged prediction. The final result after split/merge detection is shown in Figure 7.8.

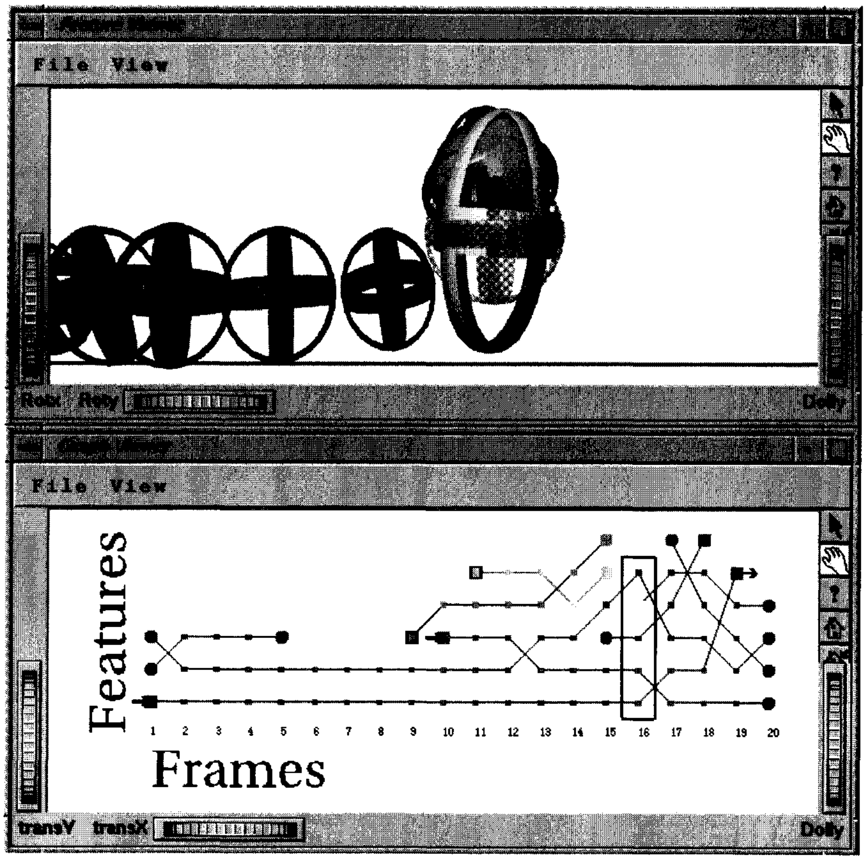


Figure 7.7: The detection of a merge event between two ending paths and a starting path. The paths are shown as opaque objects and the merged prediction is shown transparently.

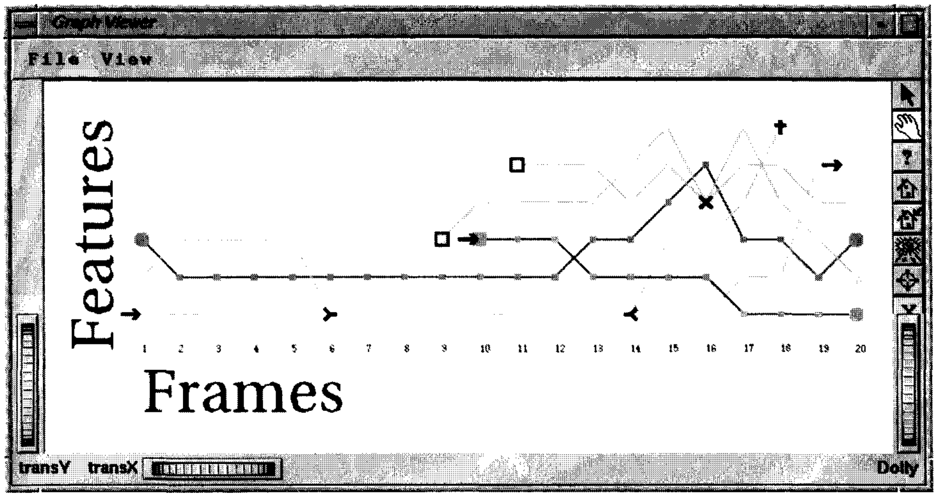


Figure 7.8: The resulting event graph after the detection of split/merge events.

7.5 Topological Events

When skeleton data is available, we have information about the topology of the features which can be used to detect topological events. A topological event can be defined as a change in the topology of a feature. Thus, the topological events may be put in the category of 'unusual changes' in the attributes of a feature. The feature continues, but something significant happens to the attribute values.

A change in topology of an object can be detected by running over the path of the object and testing the topology of successive features in the path. A change in the topology can be detected by the following criteria:

1. Counting the number of topological nodes in the skeleton graph. The number of end nodes, junction node, and loop nodes should be equal in two successive features F_1 and F_2, if the topology did not change:

$$T_{func}(F_1, F_2) = \begin{cases} \#end_1 = \#end_2 \\ \wedge \\ \#junction_1 = \#junction_2 \\ \wedge \\ \#loop_1 = \#loop_2 \end{cases} \tag{7.18}$$

   If $T_{func}(F_1, F_2)$ is false, it is certain a topological event occurred. However, the opposite is not true. In that case, we need additional tests.

2. Counting the edge-types in the topological graph. An edge-type is determined by the type of nodes it is connecting. Edges in the topological

graph may connect two end nodes (end-end), an end node to a junction node (end-junction), a junction node to a junction node (junction-junction), etcetera. If the topology of two skeleton graph is not different, per edge-type the count should be the same:

$$T_{func}(F_1, F_2) = \begin{cases} \#(end - end)_1 = \#(end - end)_2 \\ \wedge \\ \#(end - junction)_1 = \#(end - junction)_2 \\ \wedge \\ \#(end - loop)_1 = \#(end - loop)_2 \\ \wedge \\ \#(junction - loop)_1 = \#(junction - loop)_2 \\ \wedge \\ \#(junction - junction)_1 = \#(junction - junction)_2 \end{cases} \tag{7.19}$$

Again, if $T_{func}(F_1, F_2)$ is false, it is certain a topological event occurred. However, the opposite is not true. In that case, we need additional tests.

3. Running recursively over the edges. If one runs over the edges of the skeleton graph, the order in which the edge-types appear should be equal in both graphs if the topology is the same.

The first two tests are very fast. If one of them is not satisfied we know a topology event is apparent and we need not to test the third.

With the above criteria we can test if the topology of the object changed between two successive frames. If the topology changes we can use the Euler formula to characterize the change further. Section 5.4.1 showed that the number of loops can be determined by:

$$H = 1 - V + E \tag{7.20}$$

with V the number of vertices or nodes, E the number of edges, and H the number of holes in the topological graph. If H is different, then a topo-loop event is detected. If H is equal but E is different, then a topo-junction event is detected.

7.5.1 Topo-loop event

A topo-loop event can be detected by counting the number of holes in the topological graph of the features in successive frames. The number of holes is determined by the Euler formula in equation 7.20. Figure 7.9 shows an object in three successive frames. A loop is present in the second frame, but not in the other frames. This means a topo-loop event is detected.

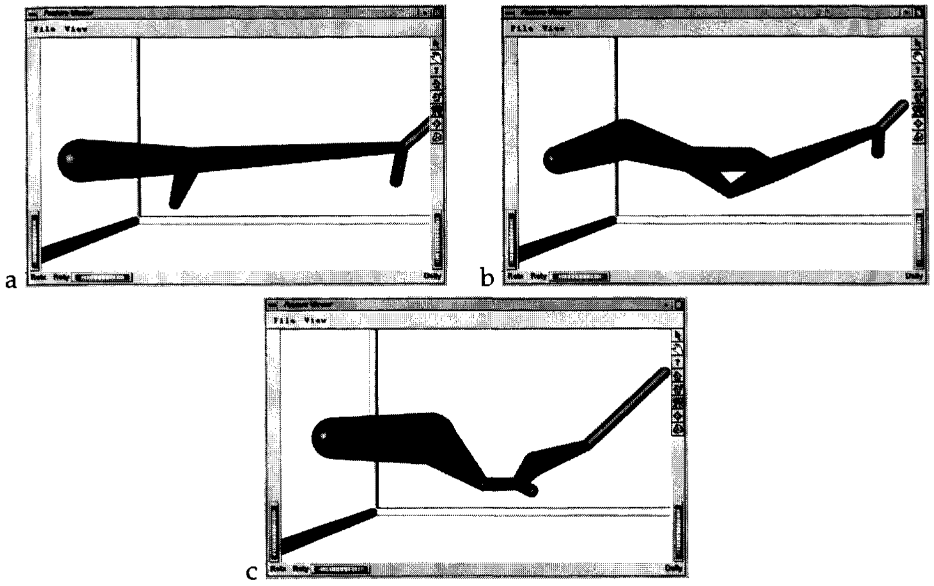


Figure 7.9: The detection of a topo-loop event. The topology of the object changes over three successive frames, the middle frame shows a loop.

7.5.2 Topo-junction event

A topo-junction event can be detected by counting the number of edges in the topological graph of the features in successive frames. Figure 7.10 shows an object in two successive frames. A junction is present in the first frame, but not in the second frame. Hence, a topo-junction event is detected.

7.6 Unresolved

Unresolved are all features of which the evolution is not explained in one or both time-directions. For instance, an unexplained ending of a path is unresolved although a correspondence is found in one direction. An unmatched feature has no correspondence in either direction and is double unresolved.

Unresolved feature events result from the failure to find a correspondence. This may be caused by a variety of reasons:

- The object may have been missed during the feature extraction phase, so the object does not exist in this frame. This happens when features are weak; i.e. either the data lies close to the selection threshold, or the number of selected nodes in the cluster is close to the cluster threshold. This

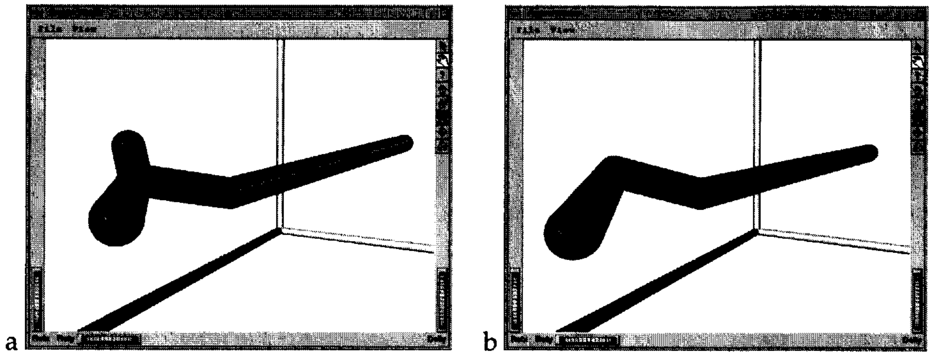


Figure 7.10: The detection of a topo-junction event. The topology of the object changes between two successive frames, a junction disappears.

may cause the object to flash on and off. We call this effect 'popping', and results in features that cannot be tracked by our method.

- The tracking tolerances may be too strict, they should be increased. However, this may also give rise to false correspondences that should not have been detected. The tolerances cannot be raised too much, as the risk of finding unwanted correspondences (false positives) will become too high.
- The time-sampling was too sparse. The objects change too much between successive frames, i.e. their behavior becomes unpredictable. The objects should exist for at least the minimal path length before participating in an event.
- The particular event cannot be detected yet. The event detection method should be extended with a new event detection algorithm.

We may put a lot of effort to solve these problems, but a number of features and events will always remain unresolved.

Using the number of unresolved features, we can calculate a solving percentage P_{solved} for the event graph:

$$P_{solved} = \frac{2N_{ftrs} - N_{unres}}{2N_{ftrs}} \times 100\% \tag{7.21}$$

where N_{ftrs} is the total number of features over all frames, and N_{unres} is the number of unresolved events. The number of features is multiplied by two, because each feature has to be corresponded in two time directions. Furthermore, the unresolved end points of paths at the first and the last frames of the time series are not counted as unresolved. In a sense these are the temporal equivalents of the spatial Entry/Exit events: they leave the time-interval. Thus, they do not count as unresolved and do not influence the solving percentage.

7.7 Visualization of Events

The event graph is a good way to visualize the events, because it visualizes the relations between features in successive frames. A merge can be easily detected as a node with more than one incoming edge, and a split has more than one outgoing edge. However, an event is not always characterized by the number of incoming and outgoing edges. In order to enhance the visualization of events the nodes in the graph are drawn with icons related to the specific type of event happening to the underlying feature.

Figure 7.11 shows the icons and the related event for all types of events that we can detect. The icons are clear, intuitive, and easy to distinguish. By visualizing the continuations as relatively small squares, the focus is automatically directed towards the special events.

▪	Continuation
●	Unresolved
→■	Entry
■→	Exit
■	Birth
■	Death
■	Merge
■	Split
■	Merge and Split
■	Topo Junction
■	Topo Loop

Figure 7.11: Icons used to visualize the different events.

Figures 6.2 and 7.4 showed intermediate results of the feature tracking and event detection process that lead to the final result shown in Figure 7.8. This result can also be drawn in a fashion that clearly distinguishes the different paths, by minimizing the number of edge crossings in the graph (Figure 7.12).

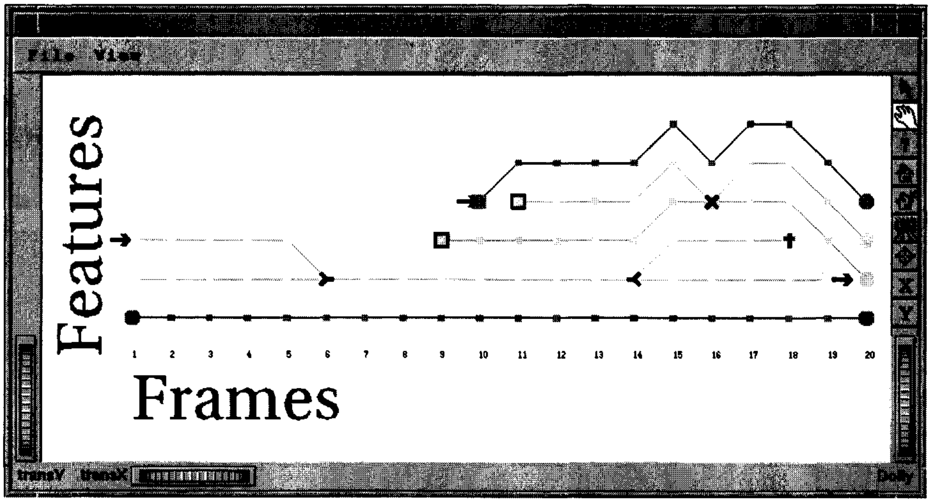


Figure 7.12: The final event graph drawn with minimal edge crossings.

7.8 Future Prospects

In the future we can extend the list of possible events by a number of new event types:

- **Bounce.** Two features collide and bounce. For example, two bouncing balls.
- **Transition.** A transition occurs from one type of feature to another type of feature (or several other types). For example, regions with high shear are thought to precede the turbulent vortex structures (of Section 8.3).
- **Popping** or **Flashing on and off.** A feature pops or flashes on/off. It is present for just one frame, or it disappears for just one frame. The latter can be detected by creating a prediction to frame $i + 2$ and test candidates in that frame.
- **Periodic patterns.** The attributes of an object behave in a periodic pattern, or the lifespan of objects behaves in a periodic patterns. For example, the Von Karman vortices in the wake of a cylinder (see Section 8.4).

For each new type of event a number of rules should be translated to specific event functions that can be tested.

The visualization of events can be enhanced. It is possible to show the nodes where a certain event occurs, i.e. *event querying*. We ask the event graph to highlight all the merge events, or topo-loop events. Furthermore, the occurrence of many splits and merges may result in a path that connects many nodes in the event graph. Since all these nodes have the same object id, they will have the same color. In this case it may be useful to have the option to draw the branches in the path with different colors.

Chapter 8

Applications

The applications serve to show the usability and performance of the visualization techniques described in this thesis. They show that feature-based visualization is effective for the analysis of large data sets, such as time-dependent data sets. The visualization of features and the analysis of their evolution, helps in gaining more insights in the physics of the application. Moreover, the applications are a good benchmark for the performance of the techniques.

The techniques are applied to four different cases:

- Data from the Pioneer Venus OCPP
- Synthetic scalar data
- CFD-data with turbulent vortex structures
- CFD-data of a flow past a tapered cylinder

The following sections describe for each application: the background, the data, the techniques applied, and the results.

8.1 Pioneer Venus OCPP

The first application concerns the data from the Pioneer Venus Orbiter that circled in orbit around the planet Venus for almost twelve years. Among the 17 instruments aboard was the Orbiter Cloud Photo Polarimeter, or OCPP. The instrument measured the intensity and polarization of sunlight reflected by the atmosphere. Unpolarized sunlight is absorbed and scattered by cloud particles, the intensity and polarization characteristics of the reflected light depend on the structure of the atmosphere.

The principal objective of the Pioneer investigation was to determine the properties of the clouds and haze, including the vertical and horizontal distribution of the particles, cloud particle size and refractive index. Also, the variations in cloud morphology and the nature of the cloud motions were investigated. Various studies ([Rossow *et al.*, 1980; Rossow *et al.*, 1990; Toigo *et al.*, 1994; Smith & Gierasch, 1996]) showed that clouds move with an average global circulation periodicity of 4.5 days.

The OCPP data is difficult to access and interpret. A first problem arises from the fact that the data is measured, and contains noise and gaps; it needs to be preprocessed (re-sampled and filtered) before it can be visualized. A second problem is that the data set is large and complex. It consists of many irregular samples in time, with for each sample a number of quantities which depend very much on the measurement conditions. As a result, it is difficult to find interesting features such as clouds.

Considering the large amount and the complexity of the data, data interpretation and visualization techniques are essential. We have created a number of tools for the access and preprocessing of the OCPP data. With these tools, we can browse through the data, and interactively explore the data in search of interesting phenomena. Furthermore, we can use the feature extraction techniques to automatically extract cloud features. The goal of the visualizations is to find a series of consecutive time samples with clouds that move consistently, and use these to verify the global circulation periodicity of 4.5 days. The research described here was also published in [Reinders *et al.*, 1997].

8.1.1 The OCPP data

The orbiter, rotating around its axis, measured the data in scan lines across the surface as illustrated in Figure 8.1a. During every scan line one of the four different wavelengths (270, 365, 550 and 935*nm*) of the scattered sunlight is measured. Thus, the surface is scanned by about 50 scan lines for every wavelength. One measurement of the planet surface is called a *map*. The 2Gb of data from the OCPP is organized in 4000 of these maps that contain collections of 2D spherical coordinates (the radius is assumed to be fixed) and the measured quantities of the scattered light.

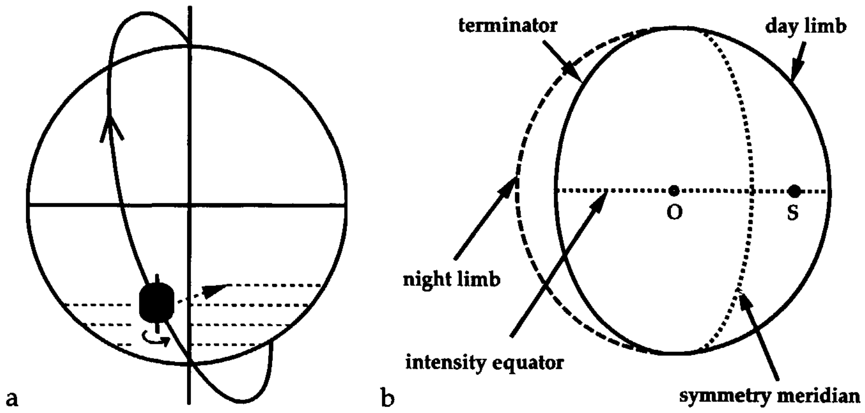


Figure 8.1: Diagrams of Venus and the Pioneer Venus Orbiter, a) scanning over the surface and b) measurement situation.

The measured properties of the reflected light are: the intensity I, the degree of linear polarization $|P|$ and the direction of polarization χ relative to the local scattering plane. From this data, we can calculate the Stokes parameters I, Q, U, and V. Furthermore, the positions of the scan points, the sub-solar point and the sub-orbiter point are given in spherical coordinates. The coordinate system is fixed in space, with the positive x-axis pointing in the direction of the constellation Aries, i.e. it does not rotate with Venus.

In Figure 8.1b a diagram of the situation is given as seen from the orbiter's point of view. In this figure O is the sub-orbiter point and S is the sub-solar point. The visible hemisphere has as its center at point O, while the illuminated hemisphere has point S as its center. These two hemispheres overlap in a symmetric segment on the planet's surface, bounded at one side by the day limb and at the other side by the terminator. The meridian which cuts the overlapping segment into equal halves is called the *symmetry meridian*. The line through O and S is called the *intensity equator*. Both are important symmetry axes [Hovenier, 1970].

The local scattering geometry (Figure 8.2) is determined by the positions of the sun and the orbiter: μ_0 is the cosine of the angle between the incoming light and the normal, μ is the cosine of the angle between the outgoing light and the normal, the azimuth angle $\phi - \phi_0$ is the angle between the plane of incidence and the plane of reflection and the phase angle θ is the angle between the incoming and outgoing direction of light.

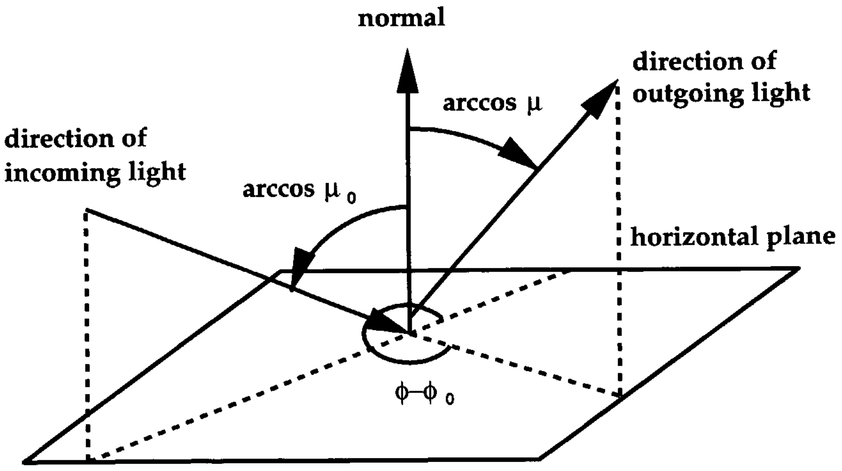


Figure 8.2: Local scattering geometry.

Table 8.1 gives an overview of the data. The scattering properties are important because they differ for each map, and may have a large effect on the measured quantities. For instance, the polarization degree depends on the phase angle, causing the data range in $|P|$ to be considerably different between two successive maps. Furthermore, the time between two maps varies considerably; there are periods of months without any measurements.

wavelength	270	365	550	935
data	I	$\|P\|$	χ	
position	longitude		latitude	
scattering properties	μ_0	μ	θ	$\phi - \phi_0$
Stokes parameters	I	Q	U	V

Table 8.1: Overview of the data measured by the Pioneer Venus OCPP.

It will be clear that the OCPP data is complex and hard to interpret. Therefore, it is important to be able to browse through the data, select specific maps, wavelengths and data quantities. We created special routines to read the OCPP data from disk, and access any desired quantity.

The following paragraphs discuss a number of preprocessing techniques that are needed before the data can be visualized. Some of the descriptions are very detailed, and may be hard to comprehend. While the preprocessing techniques are supplementary, and not really essential to understand the extraction and tracking of cloud features, the reader is free to skip to Section 8.1.2 at any time.

Re-sampling

Since the data was measured in scan lines at non-uniform intervals, it is delivered as scattered data. In order to obtain the data on a regular grid, the data is re-sampled. This is achieved by a nearest-neighbor interpolation algorithm as illustrated in Figure 8.3. The area around each grid point is divided into a number of sectors. For each sector the nearest raw data point, within a certain *search radius*, is determined. If for all sectors the nearest neighbor is found, the interpolation takes place by weighting over these points by the inverse of their distance.

Experience shows that a good choice for the number of sectors is three or four. The search radius may not be too small: this means loss of data when the distance between raw data points is larger. However, too large means searching in a large area, which means more computation time, while the effects are small. Experience shows that a good choice for the search radius is two or three times the average distance between the raw data points.

After re-sampling, the data is defined on a regular grid, which has the advantage that it can be visualized more easily, and that filtering techniques can be applied. Moreover, the feature extraction techniques discussed in this thesis can only be applied to structured data sets.

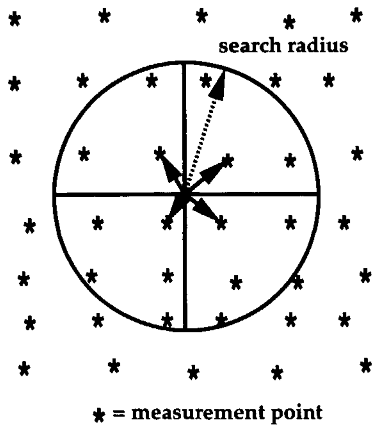


Figure 8.3: Re-sampling the raw OCPP data to a regular grid.

Limb darkening correction

When the intensity data is investigated, it becomes apparent that the intensity decreases near the limbs. This effect is called *limb darkening*, and is caused by the geometry of the planet. A spherical object reflects the light with a certain attenuation: the intensity is strongest in the middle and weaker near the edges. This is visible in the brightness distribution at wavelength $\lambda = 550nm$, see Figure 8.4. For the higher wavelengths (550 and $935nm$) scattering by gas molecules is less significant; therefore the influence of atmospheric properties on the intensity is small. However, the limb darkening also affects these lower wavelengths. This causes features near the limbs to be blurred or even become invisible.

The features near the limbs can be enhanced by a correction for the limb darkening effect based on Lambert's law. This law is based on the assumption that the light is reflected with equal intensity in all directions (perfectly diffuse reflection). This leads to the following correction procedure ([Rossow *et al.*, 1980]).

First, a disk integrated scaling factor I_d is calculated:

$$I_d = \sum_n I_n^r [\sum_n \mu_{0n}]^{-1} \tag{8.1}$$

where I_n^r is the raw intensity field at point n. Then, the appropriate zenith angle is subtracted from the raw intensity I^r:

$$I_n^c = I_n^r - I_d \mu_{0n} \tag{8.2}$$

where I_n^c is the correction field at point n. Thus, the corrected intensities are an indication of intrinsic brightness deviations from the mean brightness over

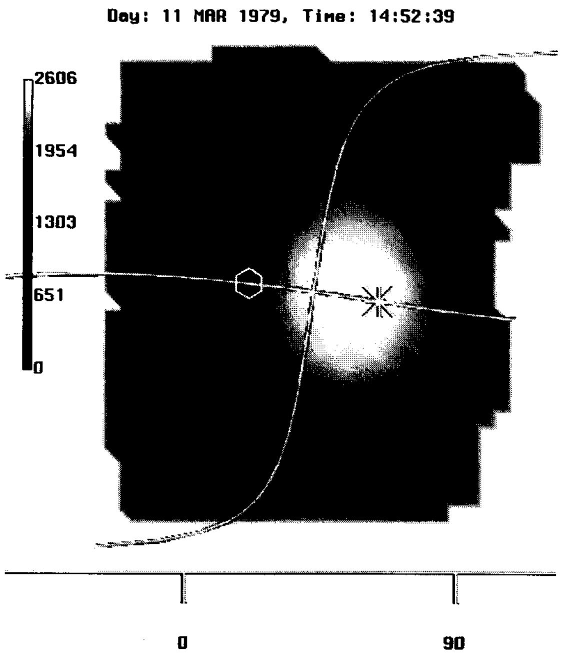


Figure 8.4: The limb darkening effect: the brightness distribution at wavelength $\lambda = 550nm$.

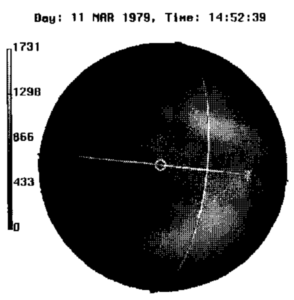
the disk. They can have positive or negative values, corresponding to relatively bright or dark features. This correction can be performed on the raw intensity data (i.e. before the re-sampling step).

8.1.2 Visualization of the OCPP data

A color map presentation of the data can be given in two ways. Since the data is available in spherical coordinates, with a constant radius, it can be presented as a 2D map. The longitude ranges from -180 to 180 degrees, and the latitude ranges from -90 to 90 degrees. However, the data will be horizontally stretched near the poles, which also deforms feature patterns. This can be avoided by projecting the data on a sphere.

Figure 8.5 presents both representations. The colors indicate the values of the scalar data. The time and date of the measurement are shown in the header. The positions of the sub orbiter and sub solar point are marked by o and * respectively, and the two symmetry axes are visualized by lines.

The correction of the intensity data for the limb darkening effect enhances the intensity patterns. A better distinction can be made between dark and light a



b

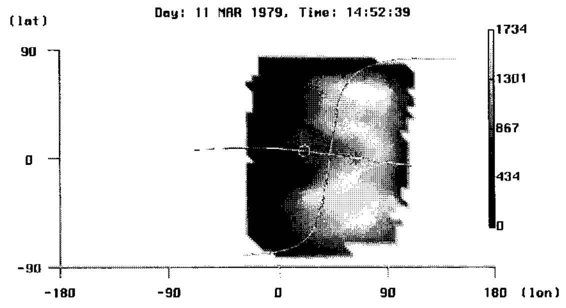


Figure 8.5: Two types of data presentation, day 169, $\lambda = 365nm$, a) in cartesian coordinates and b) in spherical coordinates.

features. The browser can be used to search significant patterns or features in the data by visual inspection. These features can then be detected automatically and visualized as described in the next section.

8.1.3 Tracking OCPP cloud features

Cloud features

The features extracted from the OCPP data are cloud formations seen in the intensity and in the degree of polarization. These cloud features can be extracted by the feature extraction pipeline discussed in Section 2.4 which uses a selection criterion.

We want a selection criterion that is invariant for different maps. Generally speaking, a cloud feature is defined as a deviation from the average value. A region with values below the surrounding is a 'dark' feature and a region with higher values is a 'light' feature. Obviously, statistical parameters such as the mean or the standard deviation (SD) are useful. If the distribution of the data is Gaussian, two thirds of the data values lie within one SD and 95% lies within 2 SD's from the mean. In both cases the criterion for selection is invariant for different maps.

Here, another advantage of the limb darkening correction arises: the correction creates approximately a Gaussian distribution. The relatively dark features have negative values and the bright features have positive values. The expression below selects dark cloud features with a deviation of 1 SD from the corrected intensity field, labeled **I365**, at wavelength $\lambda = 365nm$. The selection expression is as follows:

ranged	=	(**I365** - avg(**I365**)) / sd(**I365**)
threshold	=	1.0
select	=	**ranged** < -threshold

The simplest way to visualize the position and the size of a cloud feature is with a 2D ellipse icon. A 2D ellipse is described by an attribute set of five parameters. In Figure 8.6 a selection is made from intensity data with a wavelength $\lambda = 365nm$. The intensity data is corrected and the selection is made by the expression described above, i.e. dark features are extracted. The selected points are marked by small white crosses and the ellipses visualize the cloud features.

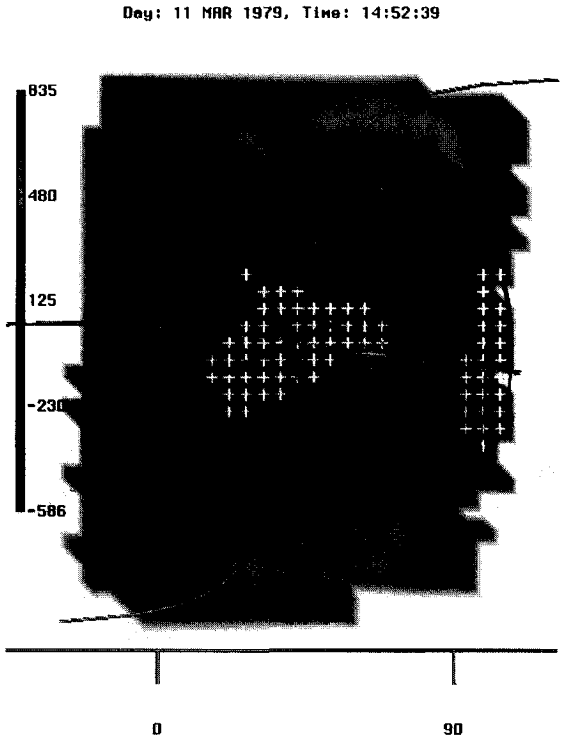


Figure 8.6: Iconic representation of the selected cloud features by ellipse fitting.

Cloud tracking

Next, we want to find a series of maps with coherently moving cloud features. We first make a rough selection from all maps. The following two restrictions are applied: the time between two successive images must be less than 12 hours and the phase angle θ must be less than 80^o. The first restriction is based on the expected speed of motion (4.5 days circulation), and the size of the visible part. Half the planet's surface is visible and this part becomes smaller with increasing phase angle; therefore the second criterion is introduced. These two criteria reduce the number of useful maps to a small number of series of maps.

The remaining maps are then explored for coherently moving features using the browser. Intensity and polarization data may show different features, so both types of data are explored. Correspondence between two cloud features in two frames is established manually, based on the following criteria:

- Position.
  The feature in the second frame must be at the position expected from the position and velocity (known in advance) of the feature in the first frame.
- Volume (or surface area).
  The two features must have approximately the same size.

Note that these are the basic feature attributes, also used for algorithmic tracking (see Section 6.2). The velocity of the cloud features can be estimated from the center positions and the time interval between two frames.

The series shown in Figure 8.7 is an example of a series with coherently moving features. It covers about 28 hours and the phase angle varies between $63.2 \leq \theta \leq 68.8$. Since the phase angle is almost constant, the threshold can be taken as an absolute value. The selection is based on the polarization degree: $|P| \geq 3.0$. In the figure six frames are shown in which features move from right to left. In the first three frames two features are visible. Based on consistency of motion we can say that the left feature moves out of the image and that only the right feature is still apparent in the fourth frame, until it almost disappears in frame six.

In Figure 8.8 the longitude and the latitude of the second feature are plotted as a function of the time. In about 28.5 hours the feature has moved from longitude = 91.9 to 12.5 degrees. During this period the latitude was almost constant. This means that the global circulation time is about 5.4 days. However, a part of the feature was outside the measurement area in the beginning and at the end of the series. This affects the position of the center point and the approximation of the circulation velocity. Without using the first and last position, the new result is a circulation time of 4.4 days. This result is very well in agreement with the expected average of 4.5 days. This supports the conclusion that the extracted feature is a real moving cloud formation.

The lessons learned from the Pioneer OCCP application are that it is important to be able to browse through the data, and to be able to explore the data in search of interesting features. This is true especially for inaccessible and complex data like the OCPP data. With the tools described here, we have a browser that can read the raw data, select a specific map, wavelength, and data quantity, and view it with user-defined color maps in one of the projections.

The combination of visual inspection and automatic feature extraction is a powerful tool to explore the OCPP data. We were able to reproduce the global

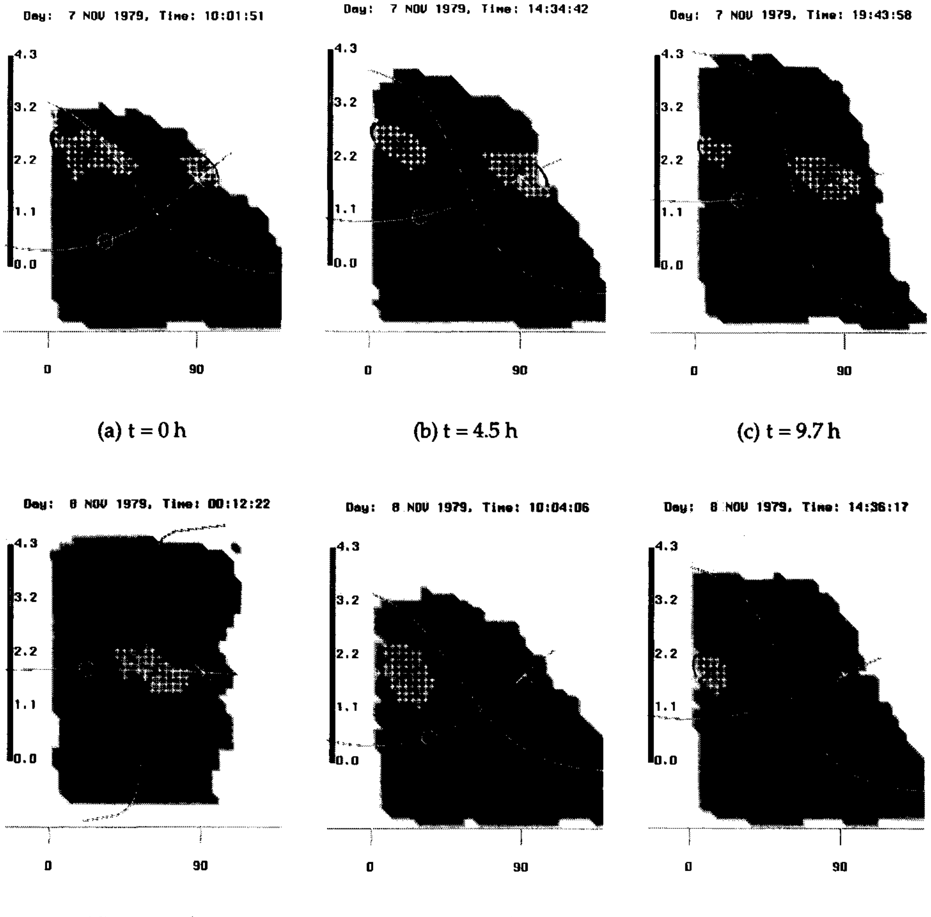


Figure 8.7: Determination of circulation velocity by means of feature tracking, day 507 – 512, $|P|$ data, $\lambda = 365nm$.

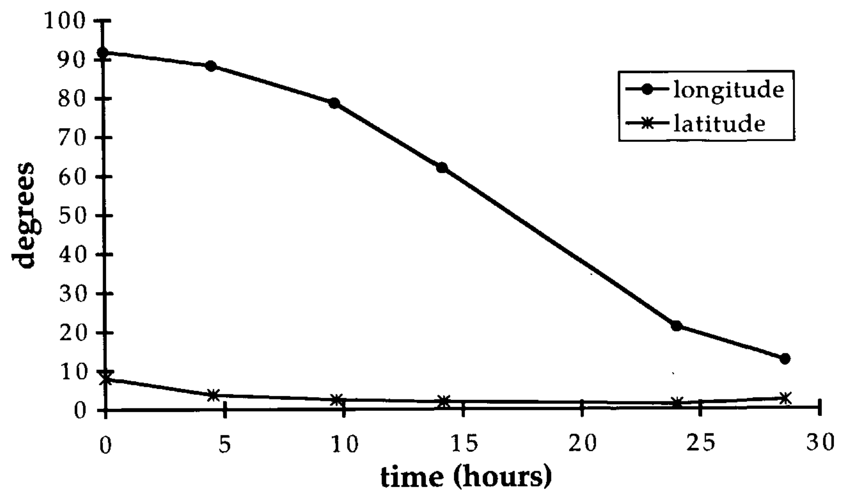


Figure 8.8: Movement of the center point of the cloud feature.

circulation periodicity clouds in 4.5 days. The example of cloud tracking consists of only two features in six frames which can be easily tracked manually. However, when tens of features in hundreds of frames are involved, it would be impossible to perform the tracking manually. This illustrates the need for automatic feature tracking.

8.2 Synthetic Data

In Chapter 4 we used synthetic data to verify the accuracy of the attribute calculation method. Here, we use the synthetic data to verify the results of the tracking procedures. This is done for two reasons: to test if the tracking algorithms find the right correspondences, and to test the performance.

Synthetic data contains known features with adjustable properties; we can generate data with features with known attributes, correspondences, and events. The feature tracking and event detection procedures should find the exact correspondences that we specified. Moreover, we can control the number of frames and the number of features per frame, which is used to test the performance of the tracking algorithm.

8.2.1 Algorithmic tests

First, we generated synthetic data with known feature evolutions in order to test the feature tracking and event detection algorithms. The a priori knowledge of feature evolutions is important and useful, because in real applications the exact correspondences are unknown. Normally, there is no objective way to determine the correctness of a correspondence, as temporal coherence is the only basis for correspondence. However, with synthetic data we can investigate the correctness of the outcome, and we can investigate the behavior of the algorithms with respect to the tracking parameters.

Synthetic data is generated by defining feature attributes at certain key-frames. The generator interpolates the feature attributes linearly between these key-frames. Then, the feature data is transformed to scalar grid data (as described in Section 4.3.1), that can be used as input for feature extraction. The resulting synthetic feature data contains features with attributes that are slightly different from the linear interpolation in the generator. Otherwise, tracking with a linear prediction scheme would be trivial.

We have generated a synthetic feature data set that has 20 frames with 80 features in total; each feature is described by an ellipsoidal fit. It is a good data set to test the tracking algorithms, because it is simple and contains all types of events. Figures 8.9 through 8.11 show the tracking results after tracking the different types of events. For each step we calculated the solved percentage P_{solved} of the graph (see equation 7.21).

Figure 8.9 is the result after tracking the continuations. The event graph shows paths with only single correspondences, no correspondences exist with multiple edges. Also, there is no explanation for the ending of paths: the nodes at the end of paths are unresolved, hence the graph is solved for 93% and not 100%.

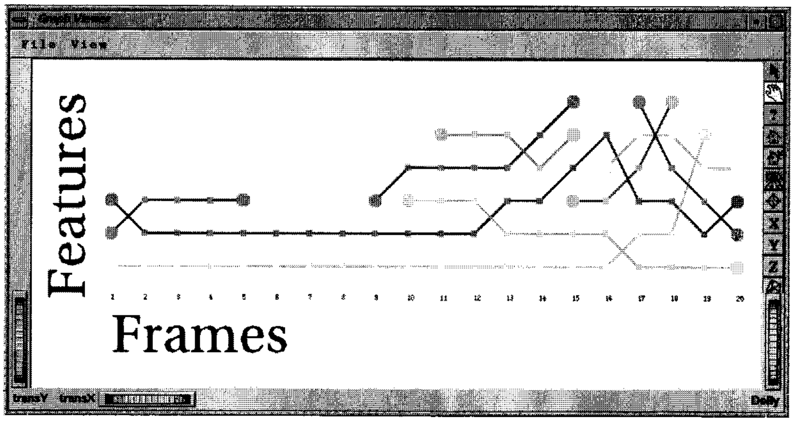


Figure 8.9: The tracking results after tracking continuations.

Figure 8.10 is the result after detecting the terminal events. A number of path endings were explained by a terminal event. However, there still remains a number of unresolved nodes, hence the graph is now solved for 96%.

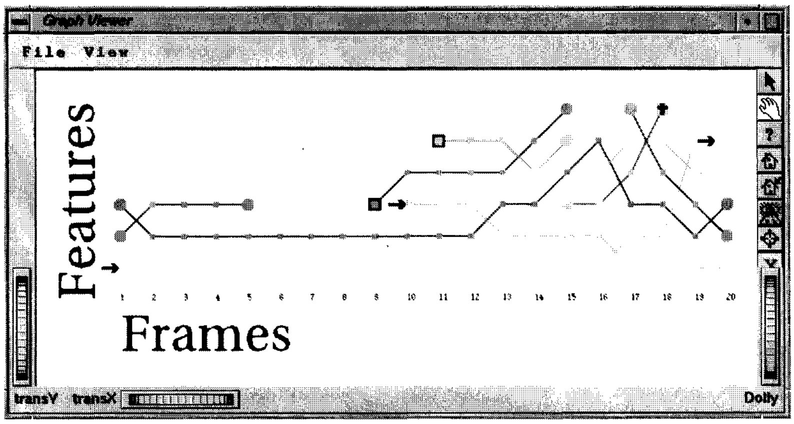


Figure 8.10: The tracking result after detecting terminal events.

Figure 8.11 is the result after detecting the split/merge events. Now there are nodes with multiple connections at the places where a split/merge event was found. Especially feature 4 in frame 16 is a special case: the incoming event is a merge and the outgoing event is a split, the object merges and splits over one frame. This is in accordance with the correspondences that were known in advance, i.e. two objects moving through each other. The graph still has unresolved nodes, however these are all at the first or the last frame of the time series, and do not count as unresolved. Hence, the graph is solved for 100% and is exactly in accordance with the known correspondences.

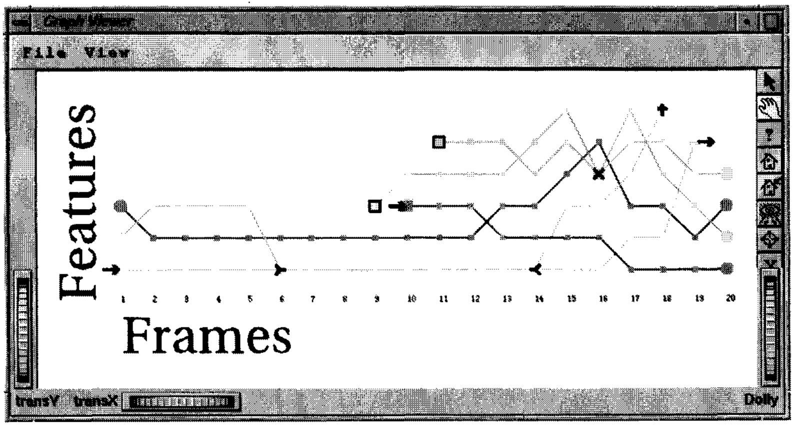


Figure 8.11: The final tracking result after detecting split/merge events.

Figure 8.12 shows the event graph with minimal crossings. It is the same graph as in Figure 8.11, but now the vertical position of the nodes is not related to the feature number, but the paths are drawn one by one. The individual paths are easily recognizable. When the number of features per frame becomes large, the graph is easier to understand.

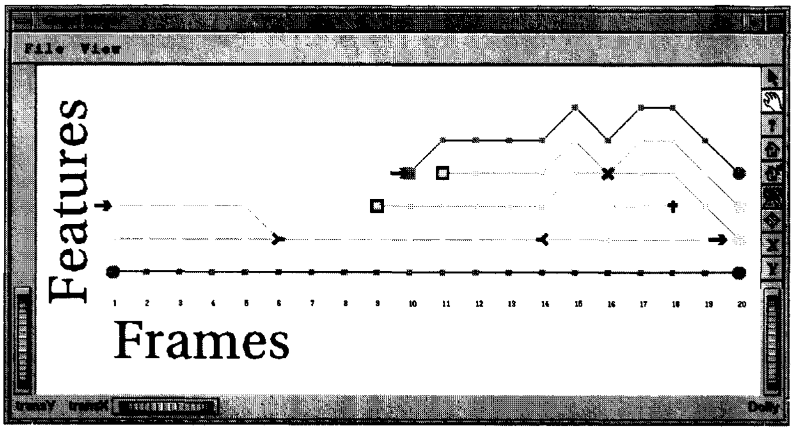


Figure 8.12: The final tracking result shown with minimal crossings.

8.2.2 Performance tests

Next, we vary the number of frames and the number of features per frame in order to test the performance of the feature tracking procedure. In Section 6.3.1, we argued that the complexity is linear with the number of frames and quadratic with the number of features per frame. We measured the user-time on an SGI Onyx with one 75 MHz R8000 processor and 256Mb of memory. The synthetic feature data in these experiments are generated without the intermediate step of grid data. They are the result of linear interpolation between features in key-frames, i.e. the tracking is trivial, and the tracking results are always correct.

Number of features per frame

We generated feature data with varying number of features per frame. The number of frames is 100 and the features are quantified by ellipsoid attribute sets. The number of features per frame is varied between 2 and 50 and the data only includes continuations, no feature interactions like split/merge. The tracking procedure is performed automatically in 4 iterations with increasing tolerances.

Figure 8.13 shows the user-time as a function of the number of features per frame. The complexity is $O(N_{ftrs}^2)$, however the profile does not immediately increase very much.

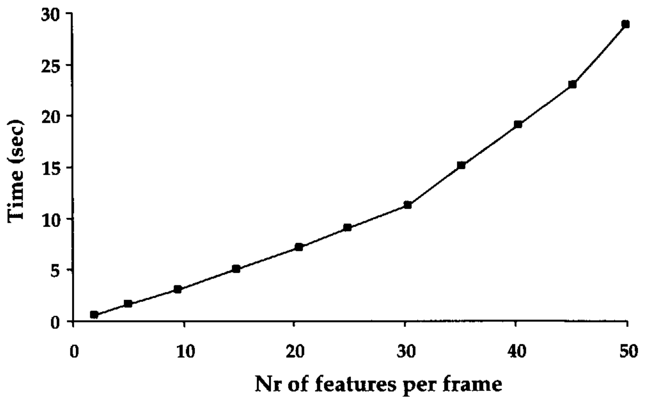


Figure 8.13: The performance of the tracking procedure as a function of the number of features per frame.

Number of frames

We generated feature data with a varying number of frames and with a fixed number of features per frame (15). The number of frames is varied from 50 to 1000 and the tracking procedure is performed automatically in 4 iterations with linearly increasing tolerances.

Figure 8.14 shows the user-time as a function of the number of frames. The complexity is almost linear (it increases a little at the end). Also, the times are small: $\leq$ 3 minutes for a considerable number of 1000 frames and features (15 features per frame).

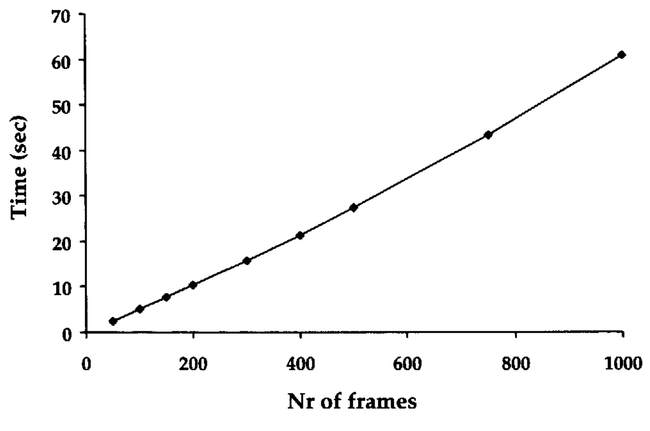


Figure 8.14: The performance of the tracking procedure as a function of the number of frames.

The lessons we learned from this application are that the algorithms for feature tracking and event detection work and find the intended results. The generation of synthetic data is a good tool to create simple data sets in order to test new algorithms. Also, the performance tests proofed the complexity to be linear with the number of frames, and quadratic with the number of features per frame. This is in accordance with our expectations.

8.3 Turbulent Vortex Structures

The next application is a CFD simulation with turbulent vortex structures. The vortex structures are line-type features with a strongly curved, tube-like shape, the so-called 'worms'. Our skeleton attribute calculation method is well suited for the description of these worms. Therefore, we used this application to test the skeleton attribute calculation. Furthermore, the data set consists of 100 frames with a large and strongly varying number of features per frame. It is a good test case for our feature tracking and event detection procedures. Finally, we are able to compare our tracking results to the results obtained by [Silver & Wang, 1997].

The data consists of one hundred time-steps with vorticity magnitude calculated at a 128^3 resolution. We obtained the original grid data as well as a tracked feature data set with the position, volume, and mass of the vortex structures[1]. Vorticity ω can be calculated from the flow velocity $\mathbf{v}$ as follows:

$$\omega = \nabla \times \mathbf{v} \tag{8.3}$$

Regions with high vorticity magnitude $\|\omega\|$ indicate the presence of vortex structures.

The turbulent vortex structures can easily be selected with a selection threshold T_{sel}: $\|\omega\| \geq T_{sel}$. After some exploration, we choose a selection threshold of $T_{sel} = 5.5$. For each selected cluster we calculated two sets of attributes: the ellipsoid and the skeleton attribute set. We found a total of 4248 features in 100 frames, and stored the feature data representation in a file that has a size of 958Kb.

8.3.1 Skeleton shape reconstruction

Figure 8.15a shows the iso-surface of the selected regions in one frame. The figure shows that the shapes of the worms are strongly curved, so that ellipsoid fitting will provide a poor approximation of the shape[2]. Large parts of

[1]Data courtesy D. Silver and X. Wang of Rutgers University, see [Silver & Wang, 1997].

[2]Still, the ellipsoid attributes are important because they provide accurate and stable global information of features, such as position and volume.

the worms fall outside the ellipsoids, as shown in Figure 8.15b. A more sophisticated method is desirable to describe the shape. For that purpose, we use skeleton attribute calculation, described in Chapter 5.

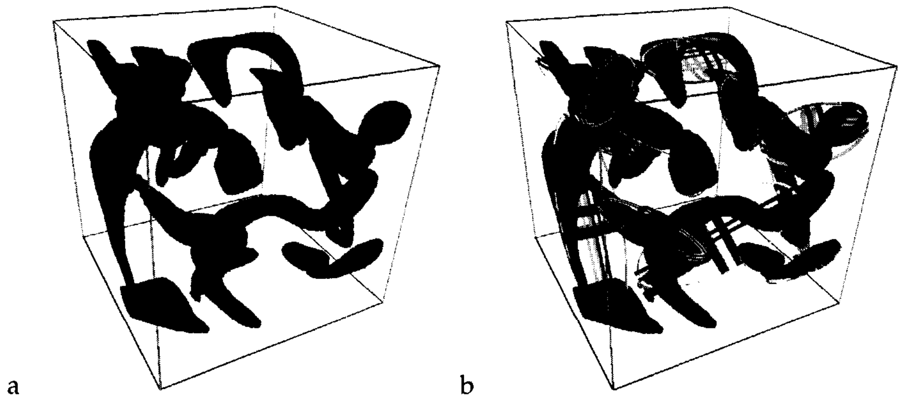


Figure 8.15: The strongly curved shape of the 'worms' are badly approximated by the ellipsoid fittings.

Figure 8.16 shows the different stages in the process of skeleton attribute calculation and shape reconstruction. First, a segmentation is made by selecting the grid nodes where the data satisfies our selection criterion, Figure 8.16a. Then, the skeleton voxels are determined by the skeletonization thinning algorithm, Figure 8.16b. These skeleton voxels are connected into a skeleton graph representation, which can be simplified ($T_{curve} = 2$ and $T_{prof} = 2$) until a small number of nodes and edges remain, Figure 8.16c. This skeleton graph is fleshed out in order to reconstruct the original shape, Figure 8.16d.

The final result of the shape reconstruction seems to be sufficiently accurate. Figure 8.17 shows the shape reconstruction created by a Hermite tube interpolation (same as Figure 8.16d) together with the surface of the segmented object (same as Figure 8.16a) shown transparently. The figure shows that the skeletons give a good impression of the structure of the vortices. The topology of the objects is reconstructed very well.

Section 5.6 showed that the skeleton approximation is inaccurate for flat shaped objects. Some of the vortex structures in Figure 8.17 are flat-shaped by nature, others are flat-shaped because they are cut by the system boundary. In both cases, the skeleton volume (equation 5.4) and the skeleton center of gravity (equation 5.5) will be inaccurate because the distance to the surface, the DT, is always the *minimal* distance.

The ellipsoid attributes, which were calculated by volume integrals, provide an accurate measure for global attributes such as position and volume (see

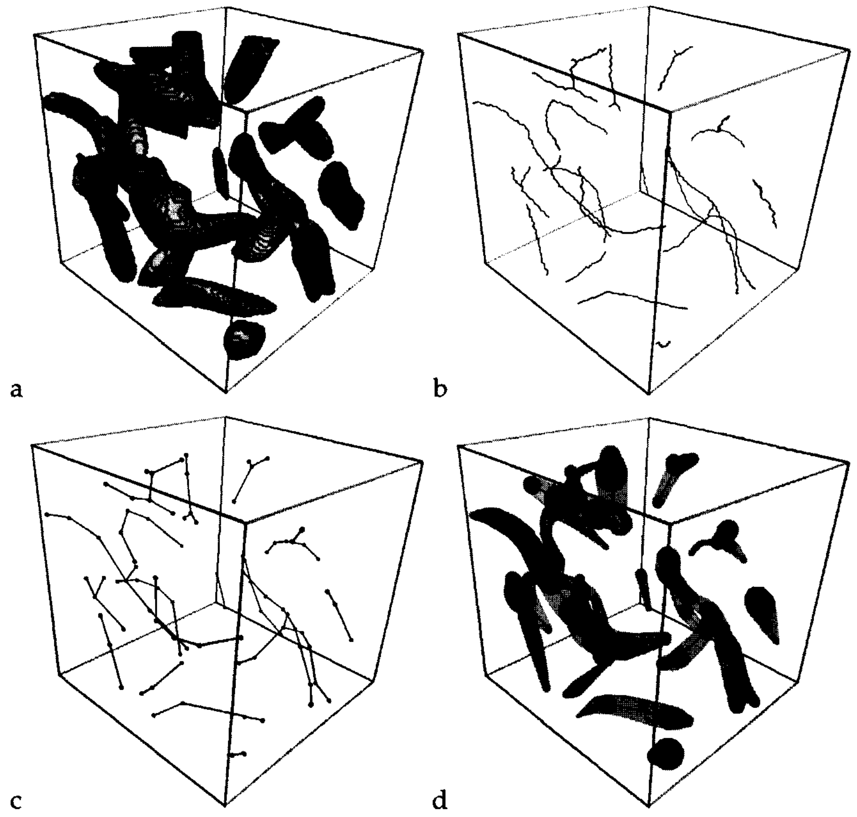


Figure 8.16: Four stages in the skeleton attribute calculation process: a) selection, b) skeletonization, c) graph simplification, and d) shape reconstruction.

Chapter 4). Therefore, we use these attributes to test the accuracy of the skeleton attributes. We compared the skeleton position and volume to the position and volume of the ellipsoid attributes.

The average distance between the ellipsoid position and skeleton position is $\|\mathbf{P}_{skel} - \mathbf{P}_{ell}\| = 1.40$ voxel. This is relatively small compared to the system dimensions of 128^3 voxels. The average volume ratio $V_{skel}/V_{ell} = 0.42$, i.e. the skeleton volume is only a fraction of 42% of the ellipsoid volume. This fraction seems unexpectedly low, but the volume depends quadratically on the diameter of the skeleton segments. Assuming the lengths of the segments are accurate, the diameter is under-estimated by 65%.

The problem with the skeleton volume arises from the fact that we use the DT as a measure for the thickness of the tubes. Possible solutions to that problem were discussed in Section 5.7. For the moment, we can only conclude that we

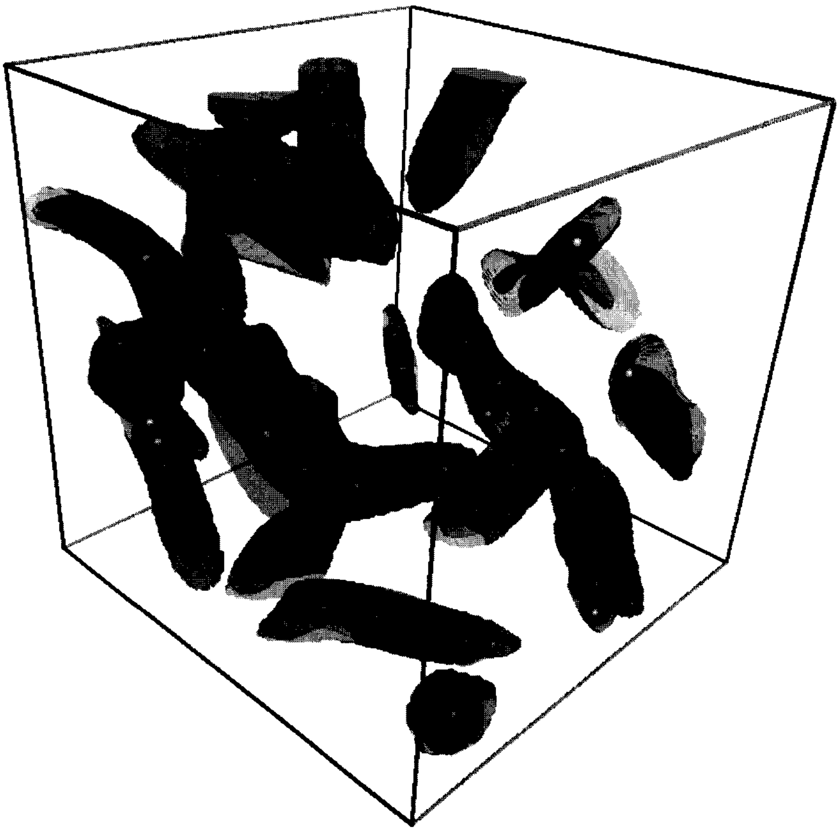


Figure 8.17: The shape reconstruction using a hermite tube interpolation together with the surface of the object shown transparently, figure is shown in color on the backside of the cover.

need to be careful, when using the skeleton volume attributes for tracking. We suggest to use volume integral quantities for tracking purposes, since they have proven to be very accurate and stable.

8.3.2 Tracking and event detection

We used the skeleton and ellipsoid attributes to track the vortex structures in the 100 frames. The average number of features per frame is relatively large: 42.5 features per frame. Also, this number varies strongly between the frames, see Figure 8.18. This means that this case study is a challenge for our tracking and event detection system. The high number of features per frame is critical for the initialization of new paths. And the strong variation in this number indicates that features evolve quickly, i.e. their lifetime is short, and they often participate in interactions. This means that feature paths may not satisfy the minimal path length criterion and many features may be left unmatched.

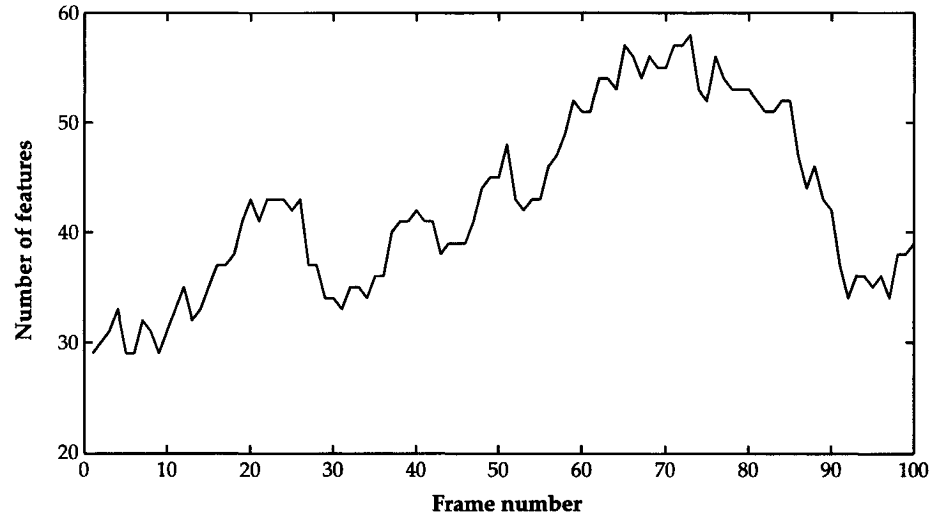


Figure 8.18: The number of features in the turbulent vortex application as a function of time.

Tracking with different correspondence functions

Section 8.3.1 showed that skeleton position and volume may be less suitable for tracking purposes. Here we test that hypothesis by comparing the tracking results using different correspondence functions. We perform feature tracking based on the volume integral attributes, the skeleton attributes, and the combination of both.

There are several event functions associated to both the skeleton and the ellipsoid attribute sets. This means that there are several possibilities to calculate the correspondence between two features. The user has to decide which correspondence function he wants to use and what tolerance and weight they should have. The available correspondence functions associated to the skeleton and ellipsoid attribute set were defined in Section 6.2.1.

In this case, we used combinations of the functions related to the ellipsoid position $\mathbf{P}_{ell}$ and volume V_{ell}, and to the skeleton position $\mathbf{P}_{skel}$, volume V_{skel}, and length L_{skel}. Table 8.2 lists these functions together with their tolerances. The weights are taken equal for each function. Some of these functions are meaningless unless they are used in combination with another; e.g. a volume test is useless without a position test. Therefore, the number of sensible combinations is limited. We tested five combinations:

1. $\mathbf{P}_{ell} + V_{ell}$
2. $\mathbf{P}_{skel} + V_{skel}$
3. $\mathbf{P}_{skel} + V_{skel} + L_{skel}$
4. $\mathbf{P}_{ell} + V_{ell} + \mathbf{P}_{skel} + V_{skel}$
5. $\mathbf{P}_{ell} + V_{ell} + \mathbf{P}_{skel} + V_{skel} + L_{skel}$

The correspondence factor resulting from each combination is calculated by equation 6.16.

Function	Tolerance
$\mathbf{P}_{ell}$	15.0
V_{ell}	0.5
$\mathbf{P}_{skel}$	15.0
V_{skel}	0.5
L_{skel}	0.5

Table 8.2: The used correspondence functions with their tolerance.

We tracked the turbulent vortex structures with each of these five combinations of correspondence functions, and compared the results. Tracking is done automatically with increasing tolerances and a minimal path length parameter set to four frames. The tolerances are increased in ten iterations until they reach the tolerances listed in Table 8.2. Then a final iteration is performed with the minimal path length set to three in order to find short paths of objects that evolve too quickly, but are consistent over three frames. We can safely do this at the end of the iterations.

Figure 8.19 shows, for each combination of correspondence functions, the percentage solved of the event graph (equation 7.21) as a function of the number of iterations. Clearly, $\mathbf{P}_{ell} + V_{ell}$ give better results than $\mathbf{P}_{skel} + V_{skel}$. The reason is that $\mathbf{P}_{skel} + V_{skel}$ uses derived attributes which may be less accurate (especially V_{skel}). Also, the addition of the L_{skel} function does not provide better results because it imposes an additional constraint, while the combination with ellipsoid functions improves the result. This shows the strength of the ellipsoid attribute data for calculating the correspondence between two features.

Event detection

The skeleton attributes do not give better results in feature tracking, but they are profitable for event detection. They allow the detection of topological events,

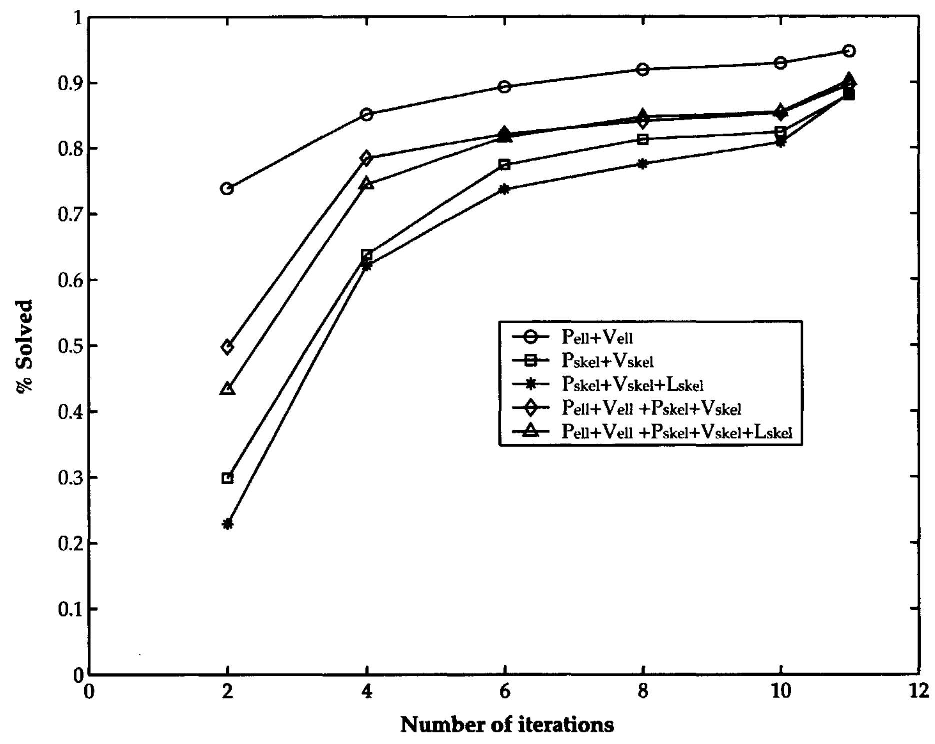


Figure 8.19: The percentage solved versus the number of iterations.

and they are beneficial for the detection of the split/merge event. Section 7.4.1 showed that skeleton data is preferred over ellipsoid data for determining a 'neighborhood' between two features. Therefore, split/merge events can best be found by using the skeleton data for the neighborhood criteria and the ellipsoid data for the correspondence criteria.

We detected the split/merge events by using the edge-to-edge distance between two skeletons with a tolerance of $T_{e-to-e} = 5.0$. The correspondence is determined by the combination of $\mathbf{P}_{ell} + V_{ell}$ with tolerances as listed in Table 8.2. We found 38 split and 23 merge events.

Then, we detected terminal events and topological events. For entry/exit events we used $\mathbf{P}_{ell}$ and for birth/death events we used V_{ell} as the criteria. Topological events were detected by the criteria described in Section 7.5. We found 82 entry, 83 exit, 75 birth, 99 death, 196 topo-junction, and 5 topo-loop events. The statistics of the final tracking results are listed in Table 8.3. The event graph is solved for 97.5%, which is a very good result.

The evolution of the turbulent vortex structures can be visualized using the

graph	count
solved	97.5%
frames	100
features	4248
paths	181
unmatched	86

event	count
continuation	3860
birth	75
death	99
entry	82
exit	83
split	38
merge	23
topo junction	196
topo loop	5
unresolved	524

Table 8.3: Statistics of the tracking and event detection results of the turbulent vortex structures application.

different skeleton icons as illustrated in Figure 5.15, or using the ellipsoid icons[3].

Comparison between two tracking methods

Next to the original grid data, we also acquired feature data with the following feature attributes: position **P**, volume V, and mass M. Furthermore, we acquired the tracking results obtained with the volume tracking method described in [Silver & Wang, 1996] which is briefly described in Section 6.7.2. In the following discussion, the solution obtained by this method is referred to as *Silver*. Since we only have position, volume, and mass, we visualize the features by spheres that show position and volume of the vortex structures.

The Silver data was transformed into our feature data and event graph representation, so that we can compare the results to our tracking method. Unfortunately, we were unable to reproduce the feature selections from the original grid data, because the exact feature extraction parameters were unknown. As a result, the tracking has to be performed on the position, volume, and mass attributes only.

The features were tracked for continuations with the parameters shown in Table 8.4, the weights are taken equal for each function. The tolerances were increased in four iterations with a minimal path length of 4, additionally one iterations is performed with a minimal path length of 3. Then, the split/merge events are detected with the neighborhood tolerance for position N_p set to 30 (twice the normal position tolerance). Finally, terminal events are detected, where we give priority to the entry/exit events. The resulting event graph is referred to as *Reinders*, and can be compared to the event graph of Silver.

[3]We created animations that can be found on the CD enclosed with this thesis or via the web: `http://www.cg.its.tudelft.nl/SciVis/Tracking`.

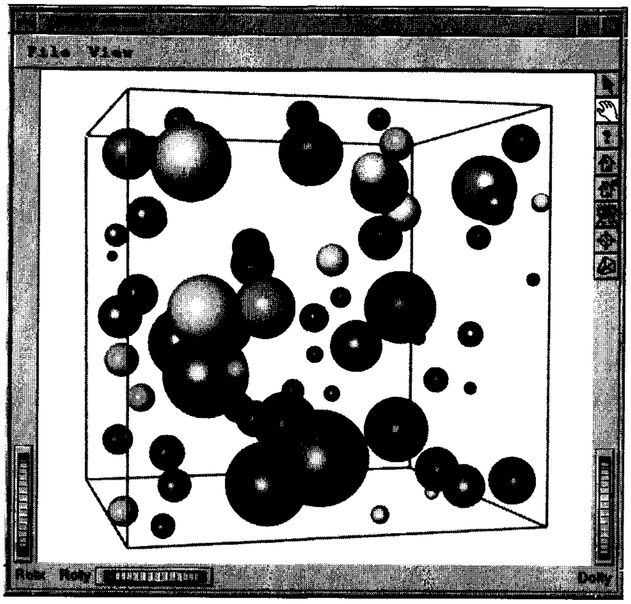


Figure 8.20: Turbulent vortex structures visualized by spheres which indicate the global position and volume of the structures.

Function	Tolerance
P	15.0
V	0.5
M	0.5
N_p	30

Table 8.4: The used event functions with their tolerance.

The two graphs can be compared by counting the number of edges that are identical and the number of edges that are different. The result is a graph correspondence factor C_g between two graphs G_1 and G_2:

$$C_g\left(G_1, G_2\right) = \frac{N_e\left(G_1 \bigcap G_2\right)}{N_e\left(G_1 \bigcup G_2\right)} \tag{8.4}$$

where $N_e(G_1 \bigcap G_2)$ is the number of edges that are present in both graphs, and $N_e(G_1 \bigcup G_2)$ is the number of edges that are present in either one graph or the other.

The comparison of the two graphs results in a graph correspondence of $C_g(\text{Reinders, Silver}) = 91.0\%$. There are 4565 coinciding edges, 295 edges in Reinders but not in Silver, and 158 edges in Silver not in Reinders. So, the two graphs are $100 \times 4565/(295 + 158 + 4565) = 91.0\%$ in agreement with each other. Table 8.5 shows the event counts for both results. The number of unmatched features is 159 in Reinders against 256 in Silver. Note that the number of exit and death events from Reinders should be added to make a fair comparison with the number of exit events from Silver. Reinders finds more continuations but less events.

	Silver	Reinders
solved	95.1%	96.6%
unmatched	257	159
paths	164	126
continuation	4431	4501
birth	205	58
entry	n/a	85
death	256	117
exit	n/a	68
split	142	105
merge	75	71
unresolved	502	345

Table 8.5: A statistical comparison of the two tracking methods by counting the number of events.

Silver finds more paths, but a number of paths have a length of two frames, with just two nodes and one edge. This is possible because the tracking is based on a two-frame algorithm. The three-frame prediction-verification scheme of Reinders results in paths with a length of at least three frames. The two frame-basis seems advantageous for detecting short, but possibly fragmented paths, while the three-frame coherence rules result in longer paths. Hence, the number of paths for Silver is larger than the number of paths for Reinders, while the number of continuations is smaller.

The large number of unmatched features by Silver suggests that tracking can be improved. Tracking by overlap misses the fast-moving, small features, while these features can be tracked by Reinders. Perhaps it is possible to enhance the method of Silver by a prediction-verification scheme.

The number of split/merge events is larger in case of Silver. This can be explained by the neighborhood criteria. Silver uses an octree representation of the features that can easily detect neighborhood by overlap, where neighborhood detection by the position attribute only may fail. The strongly curved shapes of the 'worms' are better described by a skeleton attribute set. Unfortunately, we were unable to calculate these attributes.

The difference between the two graphs can also be visualized in the graph viewer. Figure 8.21 shows the similarities and differences between the two graphs. Identical edges are shown by continuous lines, edges in Reinders not in Silver are shown by dashed lines, and edges in Silver not in Reinders are shown by dotted lines. With this tool we can examine the corresponding and deviating connections and investigate the differences between the two methods.

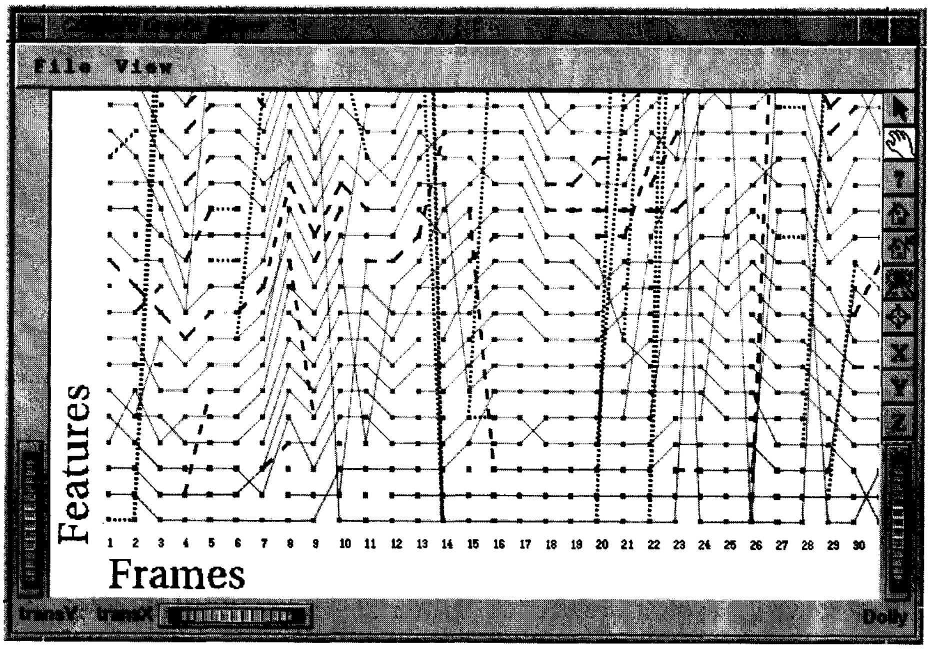


Figure 8.21: The comparison between two graphs, the continuous lines indicate edges in both graphs, the dashed lines are edges in the first (Reinders) and not in the second (Silver), and the dotted lines are edges in the second not in the first.

The application of the turbulent vortex structures shows that skeleton attributes are useful in applications were structure is important. Also, the skeletons are useful in the detection of events; they allow the detection of topological events, and they help in the detection of split/merge events. However, they are less suitable for feature tracking than ellipsoid attributes, because the derived attributes of position and volume are less accurate.

The feature tracking and event detection algorithms produce very good results. The event graph can be solved for 97.5% and 96.6%, despite the large and strongly varying number of features per frame. The results are very well (91.0%) in accordance with results obtained by a different tracking procedure. The differences may be explained, however there is no objective way to determine the correctness the two solutions.

8.4 Flow Past a Tapered Cylinder

This application is a CFD simulation of an unsteady 3D flow past a tapered cylinder, as described in [Jespersen & Levit, 1991]. The geometry of the tapered cylinder (see Figure 8.22) results in interesting 3D flow phenomena in the wake of the cylinder. Behind the cylinder unsteady vortex shedding becomes apparent. The vortices break away alternating from the left and right side of the cylinder, and have an alternating clockwise and counterclockwise rotational direction. The vortex shedding frequency f depends, among other things, on the diameter of the cylinder d which is a function of the z-coordinate.

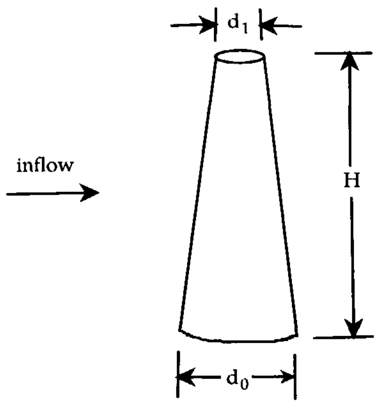


Figure 8.22: A sketch of the geometry of the tapered cylinder (taper is greatly exaggerated), taper ratio = $(d_0 - d_1)/H$.

We obtained the time-dependent data[4] of a flow past a tapered cylinder with a taper ratio of 0.01. The data is defined on a structured, cylindrical grid with $64 \times 64 \times 32$ nodes, each of which contains density d, x, y, z-momentum (xm, ym, zm), and stagnation st. The complete data set has thousands of time-steps from which we used 400, from $t = 12000$ to $t = 16000$ with an increment of 10. Each time-step is stored in a file that is 2.6 Mb large, ergo the size of the total data set we obtained is over 1.0 Gb.

Figure 8.23 shows a close-up visualization of the flow in the wake of the cylinder. The figure uses standard visualization techniques, like streamlines and colored slices, as well as feature extraction techniques; the ellipsoid icons indicate regions with high enthalpy (discussed later). The streamlines and colored slice give a nice impression of flow phenomena, however they are limited. First of all, they are two dimensional while the flow is three dimensional. Second, they do

[4]Data courtesy NASA-Ames Research Center:
`http://www.nas.nasa.gov/Research/Datasets/datasets.html.`

not give a quantification of phenomena. Third, the creation of this visualization is slow; it takes several minutes, so interactive exploration is not feasible.

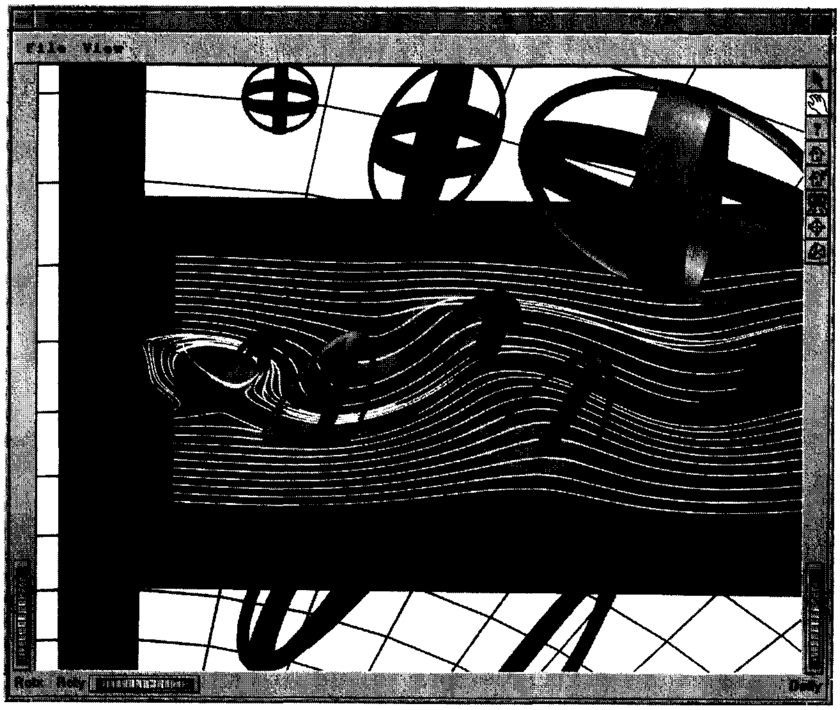


Figure 8.23: Visualization of the flow past a tapered cylinder. It shows streamlines, the colors indicate the value of enthalpy, and regions with high enthalpy are indicated by ellipsoid icons.

8.4.1 Tracking

First, we track regions with high enthalpy. The enthalpy h can be calculated from the available data values $d, xm, ym, zm,$ and st by the following equations:

$$h = \frac{\gamma}{\gamma - 1}\frac{p}{d} \tag{8.5}$$

with

$$p = (\gamma - 1)\left(st - \frac{0.5}{d}\left(xm^2 + ym^2 + zm^2\right)\right) \tag{8.6}$$

where p is the pressure and γ is the constant Gamma which is by default set to $\gamma = 1.4$. Enthalpy can be interpreted as a 'potential for heat'. High values for enthalpy indicate that there is a local exchange of heat or energy. It is known that turbulent regions in a flow have a high exchange of energy, so regions with high enthalpy may indicate turbulence.

The regions with high enthalpy are selected by the selection expression: $h \geq -0.6997$. An ellipsoidal fit is calculated for each region and the attributes are stored in a feature data file with a total size of 476 Kb. This illustrates the data reduction that can be obtained by the extraction of features: from > 1.0 Gb to 476 Kb. While it is unfeasible to explore all the data by standard visualization, now the user can easily browse through the frames, and explore the time-dependent features in search of temporal phenomena.

An artifact is immediately visible when viewing the event graph: there are defects in the frames. Figure 8.24 shows part of the event graph. Clearly something is wrong in frame 34; only one feature is found in this frame. When we examine the original flow data of the corresponding frame, it becomes clear that something is wrong with the stagnation data values. We do not know exactly what went wrong with the numerical calculations, but the selection criterion results in one big feature that occupies the entire space. After further examination of the event graph, we find defects in the data at four different frames. We removed these frames from the feature data because they disturb the tracking process.

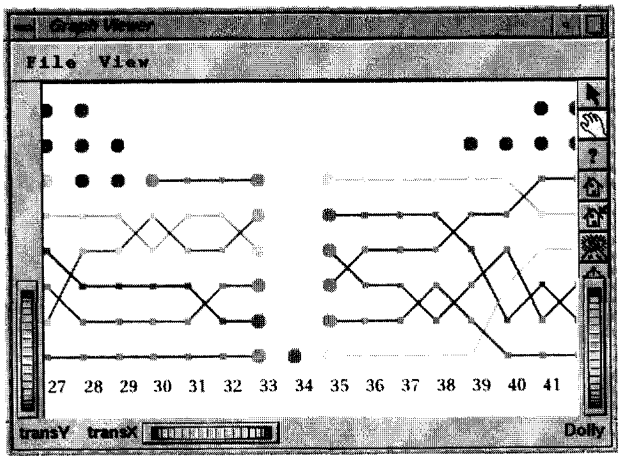


Figure 8.24: Clearly, frame 34 shows a defect in the data.

The remaining 396 frames were tracked automatically with increasing tolerances. We find a total of 4121 features (i.e. an average of 10.4 features per frame) that are highly interacting features in the wake of the cylinder. Figure 8.25 shows frame 300 in the feature viewer[5].

[5]An animation can be found on the CD enclosed with this thesis or via the web: `http://www.cg.its.tudelft.nl/SciVis/Tracking`.

	count
solved	91.3%
frames	396
features	4121
unmatched	201
paths	219
continuations	3571
births	101
deaths	64
splits	28
merges	36
unresolved	718

Table 8.6: Statistics of the enthalpy-features in the flow past a tapered cylinder.

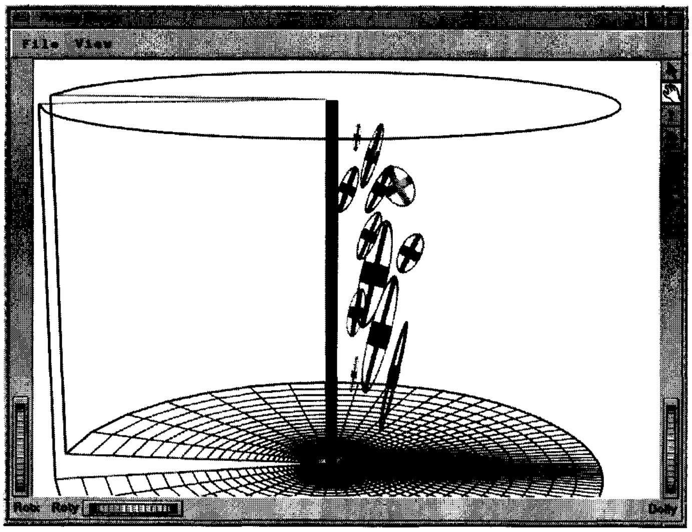


Figure 8.25: The features in frame 300 of the flow past a tapered cylinder.

This application shows that feature tracking and event detection is still interactive even for a relatively large number of frames. The user can easily experiment with the tracking parameters in order to tune the tolerances and weights of the correspondence functions for this specific application.

8.4.2 Vortex detection

This data set of a flow past a tapered cylinder was also used as a case study for the winding angle vortex detection method, which was briefly described in

Section 2.3.2. The winding angle of streamlines can be used to extract and detect vortex geometry in a 2D plane [Sadarjoen & Post, 1999; Sadarjoen & Post, 2000].

Figure 8.26 shows an example of the result obtained by this method. Streamlines are generated in a plane perpendicular to the axis of the tapered cylinder. In the wake of the cylinder a number of streamlines describe a circular path; their winding angle satisfies the selection criteria. Two clusters of streamlines are identified and for each cluster a number of attributes is calculated. The attributes are the 2D ellipse fitting parameters (position, two axes lengths, and an angle for orientation), and the rotational speed and direction. These attributes can be stored as an attribute set in our feature data representation. Finally, the attributes are mapped onto a 2D ellipse icon indicating the position and size of the vortices.

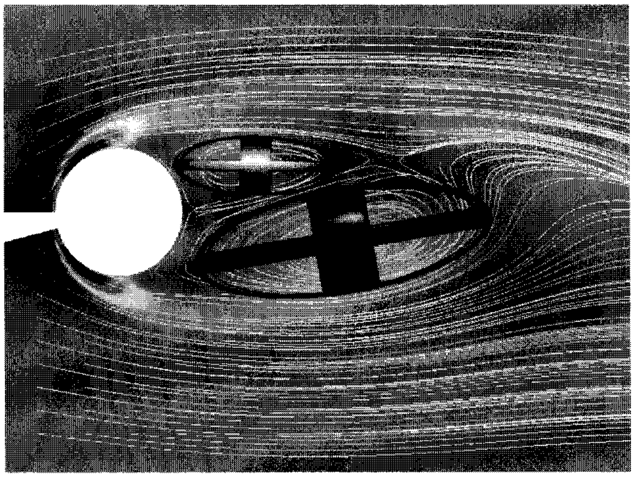


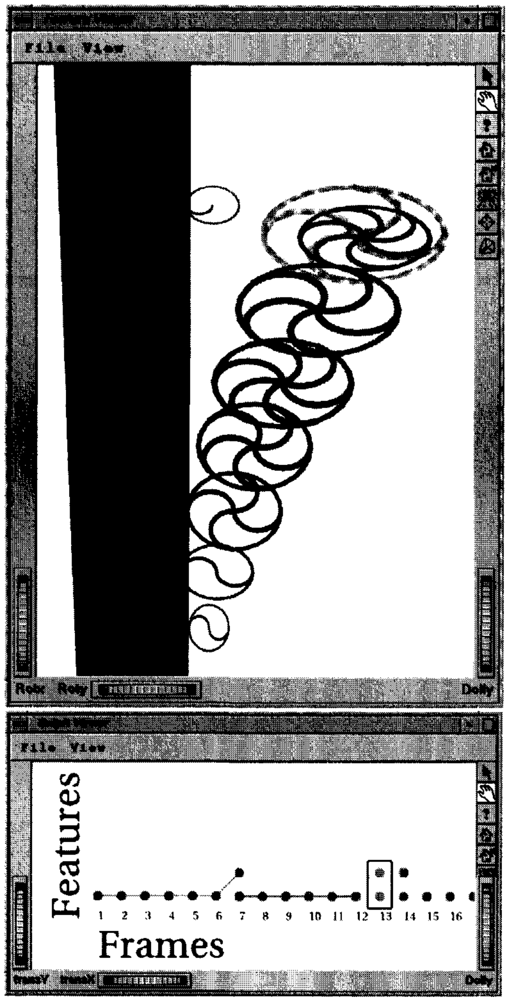
Figure 8.26: Flow past a tapered cylinder, with streamlines, vortices determined by the winding angle method, and a slice colored with λ_2. (Image courtesy Ari Sadarjoen.)

The winding angle method is essentially a two dimensional method; the winding angle can only be determined in a plane which should be chosen approximately perpendicular to the vortex core. [Sadarjoen & Post, 2000] showed that the vortices in slices perpendicular to the cylinder evolve consistently in time and therefore can be tracked. It is fairly easy to automatically track the 2D winding angle vortices in one slice over time with our tracking algorithm[6]. The result is that, in this slice, vortices appear alternating on the left and right side of the cylinder, and have alternating a clockwise and counterclockwise rotational direction. This is exactly in accordance with our expectations.

[6]An animation with 400 frames can be on the CD enclosed with this thesis, or via: `http://www.cg.its.tudelft.nl/SciVis/Tracking`.

However, our tracking algorithm can do more: it can be used to transform the 2D vortices to 3D vortices by tracking in the z-direction (parallel with the cylinder, see Figure 8.27). For one time step we can determine the 2D vortices at a number of slices with a different z-coordinate. The tracking algorithm can then be used to determine correspondences between vortices in the different slices. Once we have established these correspondences, we have obtained 3D vortices at one particular instance of time.

a



b

Figure 8.27: Tracking 2D winding angle vortices in space (z-direction) in order to obtain 3D vortices at one instance of time.

We generated streamlines in 31 slices with an increasing z-coordinate value from the bottom to the top of the cylinder (with 150 streamlines per slice). For each slice we extracted the 2D vortices with the winding angle method and stored the attributes as a frame in our feature data hierarchy. Thus, we obtained feature data with 31 frames with 2D vortex features, where each frame corresponds to a slice at the next z-level. This feature data can be tracked automatically by our tracking method.

Figure 8.27 shows one step during the feature tracking in space. Figure 8.27a shows the 2D vortices at different slices in the feature viewer. The vortex attributes are mapped onto icons with an ellipse shape, where the number of spokes indicate the rotational speed and the curve of the spokes indicates the rotational direction (the current vortex rotates clockwise). The figure shows one path of 2D vortices, the prediction at the end of the path, and two candidates in the next frame (slice). This is also shown in the event graph in Figure 8.27b, which also shows that at most two vortices are found per 2D slice, while in 3D space approximately 6 vortices exist at one instance.

The tracking in the z-direction results in a number of paths, each of which represents a vortex feature in 3D. Each path in the event graph is transformed to an attribute set with a graph representation of the 3D vortices: the vortex graph. This vortex graph is similar to the skeleton graph attribute set; it consists of nodes connected by edges. However, in this case the nodes hold 2D vortex data instead of skeleton node data. The vortex graph can be visualized by connecting the center point positions of the 2D vortices by lines according to the graph edges, see Figure 8.28. This line illustrates the approximate vortex core and the ellipse shaped icons show the vortex geometry. Thus, the 2D vortices are transformed to 3D vortex features by tracking in space.

The 3D vortices resulting from the tracking in space are features at one instance of time which in turn can be tracked in time. We repeated the process described above for 100 frames (generating a total of $100 \times 31 \times 150 = 465.000$ streamlines) and tracked the 3D vortex graphs in time. The correspondence functions associated to the vortex graph are similar to the correspondence functions of the skeleton graph (see Section 6.2.1). They are based on aggregate attributes such as the position of the center of gravity, the total volume, and the average rotational speed. Also, the graph topology can be taken into account, but we did not find any branches or loops in the vortex graphs.

The result of tracking the 3D vortices in time is remarkable: the vortices move from the top to the bottom[7]. They originate at the top of the cylinder and slowly move downward until they dissipate at the bottom. This is remarkable because the streamlines all show an *upward* motion which can expected because the size of the tapered cylinder is smallest at the top. Another remarkable phe-

[7]An animation is available on the CD-ROM enclosed with this thesis, or via: `http://www.cg.its.tudelft.nl/SciVis/Tracking`.

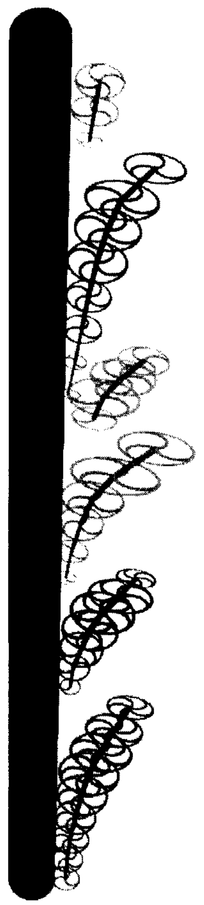


Figure 8.28: The 3D vortex graph obtained by tracking 2D vortices in space, figure is shown in color on the backside of the cover.

nomenon in the evolution is that the vortices seem to become 'weak' somewhere in the middle of the cylinder. The vortices become short (spanning less slices) and tend to show strange curves. The third vortex from the top in Figure 8.28 seems to be in this state.

The application of the flow past a tapered cylinder shows that tracking can be performed both in time and in space. Thus, we can transform 2D features to 3D features to (4D) events. Also, the application shows that tracking helps in finding artifacts such as defective frames.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This thesis covers two major directions of research in feature-based visualization: the quantification of features and the analysis of feature evolution in time-dependent data.

With respect to feature quantification, this thesis presents a new representation for feature data, an analysis of the accuracy of attribute calculation methods, and a method for the calculation of skeleton shape attributes.

With respect to the analysis of feature evolution, this thesis discusses the tracking of features in time, the detection of events in the evolution of the features, and the visualization of time-dependent phenomena.

9.1.1 Feature quantification

Chapter 3 provides a concept for a feature data representation. This representation allows the storage, manipulation, and visualization of features. A library of C^{++} -classes was developed that implements this concept. The transformation from a grid data representation to a feature data representation means a huge data reduction, as was illustrated by the applications of the turbulent vortex structures, Section 8.3, and the tapered cylinder, Section 8.4. This data reduction allows the interactive visualization and exploration of large data sets.

The class hierarchy is highly flexible; it allows all the operations on features that we need for the remainder of the research, and new classes of attributes are easily implemented. This is illustrated by the application described in Section 8.4.2 concerning the tracking of vortices. We needed to implement only two new classes for an attribute set representation: the 2D winding angle vortex attribute set and the 3D vortex graph attribute set. With these two representations we could track the 2D vortices in the z-direction, and the 3D vortices in time. The tracking process itself needed no alterations.

Chapter 4 describes an experimental approach for the analysis of the accuracy of attribute calculation methods. The accuracy of two calculation methods was investigated: the volume integrals, and the ellipsoid fitting method. Both turned out to be very accurate and stable with respect to noise.

While performing the experiments it became clear that much can be learned about the behavior of the feature extraction method. Therefore, we consider this type of experiments extremely important for the exploration and validation of visualization techniques, and we recommend to do similar studies with any new visualization method. A number of guidelines could be given for the feature extraction process that we use.

Chapter 5 presents a method for the calculation of attributes that describe the shape of a feature by means of skeletons. The skeleton attribute set provides a better description of shape compared to the crude ellipsoid fittings. The precision of the skeleton shape description can be controlled by two parameters: the curve threshold T_{curve} and the profile threshold T_{prof}. Naturally, a more precise description leads to less data reduction, but still the data reduction is significant.

The skeleton attribute set is used to create a shape reconstruction of the original shape. The skeleton attributes proved to be a reasonable quantification for the 'worms' that were found in the application of the turbulent vortex structures in Section 8.3.1. The application also showed that flat-shaped features are described less accurately. The reason is that the DT provides a minimal distance to the surface. In these cases, other descriptions for the cross-section may be useful. Nevertheless, the skeletons can be used for a reconstruction of the shape for tube-like features that have a circular cross-section.

The skeleton attributes proved to be useful in the process of event detection because they provide a topological description. New types of events are possible: the topological events. We can detect the occurrences of loops and changes in the number of junctions. Additionally, the skeletons provide a better detection of split/merge events. The neighborhood criterion can be determined more accurately, so that the tolerances in the correspondence tests can remain strict which results in less false-positives.

9.1.2 Feature evolution analysis

The extraction and quantification of features allows us to analyze the evolution of the features in time-dependent data. Features can be extracted and characterized in each frame, and the resulting attributes can be represented by our feature data representation. Using this representation we can track features in time and investigate their evolution.

Chapter 6 illustrates the tracking of features in time. Features are tracked based on a prediction-verification scheme which leads to smoothly evolving, continuous paths. The correspondences between features is obtained by attribute-correspondence, additional (grid) information is unnecessary. This is a huge advantage; the use of the feature data representation makes it possible to interactively explore the time-dependent phenomena, which is virtually impossible when grid data is used.

The user can interactively change tracking parameters, such as the tolerances and weights of the correspondence functions, and investigate the effects on the tracking results. It is feasible to perform the tracking in multiple (forward-and-backward) passes because the feature data representation is small in size. Experience shows that the best tracking results are obtained when we linearly increase the tolerances in a number of successive iterations. The tracking process will first find the obvious paths, and then the more indistinct paths.

The result of the feature tracking step is a number of continuous paths, where each path represents the motion and evolution of one object. Each feature in the path represents the instantaneous characteristics of that object. The evolution of the object's attributes can be visualized by displaying the sequence of feature attributes in the path.

Chapter 7 shows the automatic detection of significant events in the evolution of objects. Currently, the following events can be detected: continuation, birth/death, entry/exit, split/merge, and topological loop/junction events. Different types of events may be added in the future.

Event detection raises the level of abstraction to a yet higher level because it focuses on the important stages in the evolution of features. Uninteresting frames where no events occur may be skipped, or the user can jump to a frame where a particular event occurs and investigate the participating features in close detail. Thus, it is possible to create a compilation of the complete time series with only interesting episodes. This is similar to the summary of a soccer match with just the goals and other highlights of the match.

The applications show that we are able to explain over 90% of every possible connection in the event graph. Of course this result depends on the tolerances that were chosen; it is possible to solve the graph for 100% by increasing the tolerances to infinity. However, this will result in many false-positives which are undesirable. Therefore, it is essential to check the tracking results if they make sense. We did that for every application by careful visual inspection, as there are no objective criteria to verify the tracking results (except for the synthetic data sets).

Sections 6.5 and 7.7 describe the visualization of feature evolutions by means of a linked combination of two viewers: the feature viewer and the graph viewer. The feature viewer shows the features in 3D space, and the graph viewer shows the event graph.

The linked combination of the two viewers is a powerful tool for the exploration of time-dependent data. The two visualizations can be used during the tracking process in order to show intermediate results. Also, the two viewers are used for visual playback of the complete time series, selected paths, and selected time segments. Furthermore, event querying is possible: highlight only a particular type of event. Thus, we can analyze the results of the feature tracking and the event detection process.

The use of feature tracking, event detection, and visualization is a powerful tool for the analysis of feature evolution in time-dependent data. We believe that feature-based visualization is the only practical way to visually analyze large data sets, such as time-dependent data sets.

9.2 Future Work

Of course, future work includes the extension of currently available techniques. The work may be focused on the extraction and representation of new types of features, and the calculation of different types of attribute sets. The feature tracking procedure can be refined, or the detection of new types of events may be included. A discussion of such future work has been given in each corresponding chapter as future prospects. Here, we will indicate some more general long-term research topics.

9.2.1 Adaptive temporal refinement

Adaptive temporal refinement would be an important addition to the current feature tracking and event detection process. The sampling of the frames does not necessarily have to be equidistant in time. The tracking process already supports this because the prediction-verification scheme is independent of the time between frames, as the extrapolation already has a variable time base.

The time between frames may be varied in accordance with the variance of the data in time. Turbulent periods require a denser sampling in time, while periods with less turbulence can be simulated with a sparse sampling. This is similar to local (spatial) refinement of the grid. Currently, time-dependent numerical simulations often calculate many time steps in order to obtain a certain accuracy, only small part of the calculated frames is saved. Often, 90% of the data sets is discarded because they consume too much disk space, and they are not necessary for the analysis of evolutionary phenomena.

With feature tracking, it is possible to automatically recognize the episodes in the time series where interesting phenomena occur. It should be possible to restart the simulation from a key-frame at the beginning of such an episode, and to calculate new time steps with a denser sampling in time, and save more frames for analysis. Thus, we can focus on a small time-interval, and investigate the evolving phenomena in more detail. It is even possible to focus on the evolution of one feature in a small range in space, in a small period in time, i.e. to focus both in space and in time.

9.2.2 Distributed computing

Distributed computing is also an important addition to the current system. Currently, the numerical simulations are performed on super-computers, and the resulting grid data is transported to the visualization workstation where the steps of feature extraction, feature tracking, event detection, and visualization are performed.

The process can be distributed in a better way, so that computationally intensive steps are performed on the super computer. It should be possible to perform the first part of simulation, feature extraction, and feature quantification on the super computer, and then transport the feature data representation to the visualization workstation where the features can be visualized and investigated.

A practical problem that needs to be solved is the implementation of stand-alone routines for the feature extraction procedures. Currently, these procedures are implemented in the AVS visualization software package. Effort should be put in creating a stand-alone implementation of these procedures.

9.2.3 Computational steering

Once we have improved the distribution, it should be possible to perform computational steering. The user is able to change model parameters, perform the simulation and feature extraction steps remotely on a super computer, and analyze the resulting feature data on the local workstation. Model parameters include the parameters of the numerical simulation, and the feature extraction parameters. These parameters can be changed on the super computer, while the tracking parameters can be changed interactively on the visualization workstation.

Future work should focus on the relations between simulation parameters, feature extraction parameters, and feature visualization. Visualizations need to be shown while running the simulation on the background. Based on the visualizations the user may decide to change parameters and start-up new simulations. It is necessary to know our way back from the visualization to the input parameters of the simulation, and to the feature extraction parameters. This process of investigation can be controlled using a computational steering environment [Mulder *et al.*, 1999].

Although the research described in this thesis provides new tools and insights in the feature-based visualization of large data sets, there still remain many challenging ideas and visions for future research.

Bibliography

Adams, R., & Bischof, L. 1994. Seeded Region Growing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **16**(6), 641–646.

Adiv, G. 1985. Determining 3D Motion and Structure from Optical Flows Generated by several Moving Objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **7**, 384–401.

Arcelli, C., & Sanniti di Baja, G. 1985. A Width-Independent Fast Thinning Algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **7**(4), 463–474.

Arcelli, C., & Sanniti di Baja, G. 1989. A One-Pass Two-Operation Process to Detect the Skeletal Pixels on the 4-Distance Transform. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **11**(4), 411–414.

Arnaud, Y., Debois, M., & Maizi, J. 1992. Automatic Tracking and Characterization of African Convective Systems on Meteosat Pictures. *J. of Appl. Meteorology*, **31**(May), 443–453.

Atkins, M.S., & Mackiewich, B.T. 1998. Fully Automatic Segmentation of the Brain in MRI. *IEEE Trans. Medical Imaging*, **17**(1), 98–107.

Ballard, D.H., & Brown, C.M. 1982. *Computer Vision*. Prentice-Hall.

Banks, D.C., & Singer, B.A. 1995. A Predictor-Corrector Technique for Visualizing Unsteady Flow. *IEEE Trans. on Visualization and Computer Graphics*, **1**(2), 151–163.

Bartz, D., Straßer, W., Skalej, M., & Welte, D. 1999. Interactive Exploration of Extra- and Intracranial Blood Vessels. *Pages 398–392 of:* Ebert, D., Gross, M., & Hamann, B. (eds), *Proc. IEEE Visualization '99*.

Batra, R.K., & Hesselink, L. 1999 (oct). Feature Comparisons of 3D Vector Fields using Earth Mover's Distance. *Pages 105–111 of:* Ebert, D., Gross, M., & Hamann, B. (eds), *Proc. IEEE Visualization '99*.

Becker, J., & Rumpf, M. 1998. Visualization of Time-Dependent Velocity Fields by Texture Transport. *Pages 91–101 of:* Bartz, D. (ed), *Visualization in Scientific Computing '98*. Springer Verlag.

Bezdek, J.C., Hall, L.O., & Clark, L.P. 1993. Review of MR Segmentation Images using Pattern Recognition. *Medical Physics*, **20**(4), 1033–1048.

Blum, H. 1967. A Transformation for Extracting new Descriptors of Shape. *In:* Wathen-Dunn, W. (ed), *Models for the Perception of Speech and Visual Form*. MIT Press.

Borgefors, G. 1984. Distance Transformations in Arbitrary Dimensions. *Computer Vision, Graphics, and Image Processing*, **27**(3), 321–345.

Boykov, Y., & Jolly, M-P. 2000. Interactive Organ Segmentation using Graph Cuts. *Pages 276–285 of:* Delp, S.L., DiGioia, A.M., & Jaramaz, B. (eds), *Medical Image Computing and Computer-Assisted Intervention - Proc. MICCAI 2000*. Lecture Notes in Computer Science. Pittsburgh, PA, USA: Springer.

de Bruin, P.W., Vos, F.M., Vossepoel, A.M., Post, F.H., de Blok, S.B., & Aarden, A.W.J.M. 1999. Supporting Hysteroscopic Surgery by 3D Imaging, Modelling and Visualization. *Pages 150–157 of:* Boasson, M., Kaandorp, J.A., Tonino, J.F.M., & Vosselman, M.G. (eds), *Proc. ASCI '99*. Advanced School for Computing and Imaging.

Castleman, K.R. 1996. *Digital Image Processing*. Prentice-Hall.

Cignoni, P., Montani, C., & Scopigno, R. 1998. A Comparison of Mesh Simplification Algorithms. *Computers & Graphics*, **22**.

Cohen, L. 1991. On Active Contour Models and Balloons. *Computer Vision, Graphics, and Image Processing: Image Understanding*, **53**(2), 211–218.

Danielsson, P. E. 1980. Euclidean Distance Mapping. *Computer Graphics and Image Processing*, **14**, 227–248.

Delmarcelle, T., & Hesselink, L. 1992. Visualization of Second Order Tensor Fields and Matrix Data. *Pages 316–323 of:* Kaufman, A. E., & Nielson, G.M. (eds), *Proc. IEEE Visualization '92*.

Frank, K., & Lang, U. 1998. Data-Dependent Surface Simplification. *Pages 3–12 of:* Bartz, D. (ed), *Visualization in Scientific Computing '98*. Springer Verlag.

Fung, Y.C. 1965. *Foundations of Solid Mechanics*. Englewood Cliffs: Prentice Hall.

Gagvani, N., Kenchammana-Hosekote, D., & Silver, D. 1998. Volume Animation using the Skeleton Tree. *Pages 47–53 of:* Lorensen, W., & Yagel, R. (eds), *Proc. IEEE Symp. on Volume Visualization*.

Gibson, S. 1998. Constrained Elastic Surfacenets: Generating Smooth Surfaces from Binary Sampled Data. *Pages 888–898 of: Medical Image Computing and Computer-Assisted Intervention - Proc. MICCAI '98.*

Globus, A., Levit, C., & Lasinski, T. 1991. A Tool for Visualizing the Topology of Three-Dimensional Vector Fields. *Pages 33–40 of:* Nielson, G.M., & Rosenblum, L. (eds), *Proc. IEEE Visualization '91.*

in den Haak, M.D., Spoelder, H.J.W., & Groen, F.C.A. 1992. Matching of Images by Using Automatically Selected regions of Interest. *Pages 27–40 of:* Dietz, J.L.G. (ed), *Proc. CSN '92.*

Haber, R. B., & McNabb, D. A. 1990. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. *Pages 75–93 of:* Nielson, G. M., Shriver, B. D., & Rosenblum, L. (eds), *Visualization in Scientific Computing.* IEEE Computer Society Press.

Haimes, R., & Kenwright, D. 1999. On the Velocity Gradient Tensor and Fluid Feature Extraction. *In: Proc. of the 14th AIAA Computational Fluid Dynamics Conference.* American Institute of Aeronautics and Astronautics. AIAA paper 99-3291.

Helman, J.L., & Hesselink, L. 1989. Representation and Display of Vector Field Topology in Fluid Flow Data Sets. *IEEE Computer,* **22**(8), 27–36.

Helman, J.L., & Hesselink, L. 1990. Surface Representations of Two- and Three-Dimensional Separated Flows. *Pages 6–13 of: Proc. IEEE Visualization '90.*

Helman, J.L., & Hesselink, L. 1991. Visualization of Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications,* **11**(3), 36–46.

Henze, C. 1998. Feature Detection in Linked Derived Spaces. *Pages 87–94 of:* Ebert, D., Hagen, H., & Rushmeier, H. (eds), *Proc. IEEE Visualization '98.*

Hildreth, E.C. 1984. Computations Underlying the Measurement of Visual Motion. *Artificial Intelligence,* 309–354.

Hovenier, J.W. 1970. Principles of Symmetry for Polarization Studies of Planets. *Astron. & Astrophys.,* 7(1), 86–91.

Jespersen, D.C., & Levit, C. 1991 (January). *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.

Jonker, P. P., & Vossepoel, A. M. 1995. On Skeletonization Algorithms for 2, 3,.. N Dimensional Images. *Pages 71–80 of:* Dori, D., & Bruckstein, A. (eds), *Proc. Shape, Structure and Pattern Recognition '94.* Nahariya, Israel: World Scientific Singapore.

Kalivas, D.S., & Sawchuk, A.A. 1991. A Region Matching Motion Estimation Algorithm. *Computer Vision, Graphics, and Image Processing: Image Understanding*, **54**(2), 275–288.

Kass, M., Witkin, A., & Terzopoulos, D. 1988. Snakes: Active Contour Models. *Int. J. of Computer Vision*, **1**(4), 321–331.

Kenwright, D.N. 1998. Automatic Detection of Open and Closed Separation and Attachment Lines. *Pages 151–158 of*: Ebert, D., Hagen, H., & Rushmeier, H. (eds), *Proc. IEEE Visualization '98*.

Kenwright, D.N. 1999. Automatic Flow Feature Detection Techniques for Tera-Scale Data Analysis. *In*: Banks, D.C., Kenwright, D.N., Post, F.H., & Silver, D. (eds), *IEEE Visualization '99, Tutorial 6, Feature Extraction and Visualization of Time-Dependent Flow Fields*.

Kenwright, D.N., & Haimes, R. 1997. Vortex Identification - Applications in Aerodynamics: A Case Study. *Pages 413–416 of*: Yagel, R., & Hagen, H. (eds), *Proc. IEEE Visualization '97*.

Klein, R., Liebich, G., & Straßer, W. 1996. Mesh Reduction with Error Control. *Pages 311–318 of*: Yagel, R. (ed), *Proc. IEEE Visualization '96*.

Lavin, Y., Levi, Y., & Hesselink, L. 1997. Singularities in Nonuniform Tensor Fields. *Pages 59–66 of*: Yagel, R., & Hagen, H. (eds), *Proc. IEEE Visualization '97*.

Lavin, Y., Batra, R., & Hesselink, L. 1998. Feature Comparison of Vector Fields using the Earth Mover's Distance. *Pages 103–110 of*: Ebert, D., Hagen, H., & Rushmeier, H. (eds), *Proc. IEEE Visualization '98*.

Lee, T. C., Kashyap, R. L., & Chu, C. N. 1994. Building Skeleton Models via 3-D Medial Surface/Axis Thinning Algorithms. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, **56**(6), 462–478.

de Leeuw, W.C., & van Liere, R. 1999. Collapsing Flow Topology using Area Metrics. *Pages 349–354 of*: Ebert, D., Gross, M., & Hamann, B. (eds), *Proc. IEEE Visualization '99*.

Lelieveldt, B.P.F. 1999. *Anatomical Models in Cardiovascular Image Analysis*. Ph.D. thesis, Universiteit Leiden.

Leymarie, F., & Levine, M. D. 1992a. Simulating the Grassfire Transform using an Active Contour Model. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **14**(1), 56–75.

Leymarie, F., & Levine, M. D. 1992b. Tracking Deformable Objects in the Plane Using an Active Contour Model. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **15**(6), 617–634.

Lobregt, S., Verbeek, P. W., & Groen, F. C. A. 1980. Three-Dimensional Skeletonization: Principle and Algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **2**(1), 75–77.

Lorensen, W.E., & Cline, H.E. 1987. Marching Cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, **21**(4), 163–169.

Lovely, D., & Haimes, R. 1999. Shock Detection from Computational Fluid Dynamics Results. *In: Proc. of the 14th AIAA Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics. AIAA paper 99-3285.

Ma, K-L., van Rosendale, J., & Vermeer, W. 1996. 3D Shock Wave Visualization on Unstructured Grids. *Pages 87–94 of:* Yagel, R., & Nielson, G.M. (eds), *Proc. IEEE Symp. on Volume Visualization '96.*

McInerney, T., & Terzopoulos, D. 1996. Deformable Models in Medical Image Analysis: a survey. *Medical Image Analysis*, **1**(2), 91–108.

Mulder, J.D., van Wijk, J.J., & van Liere, R. 1999. A Survey of Computational Steering Environments. *Future Generation Computer Systems*, **15**(1), 119–129.

Nakajima, S., Atsumi, H., Bhalerao, A., Jolesz, F., Kikinis, R., Yoshimine, T., Moriarty, T., & Stieg, P. 1997. Computer-assisted Surgical Planning for Cerebrovascular Neurosurgery. *Neurosurgery*, **41**, 403–409.

Ogniewicz, R. L., & Kübler, O. 1995. Hierarchic Voronoi Skeletons. *Pattern Recognition*, **28**(3), 343–359.

Pagendarm, H.G., & Seitz, B. 1993. An Algorithm for Detection and Visualization of Discontinuities in Scientific Data Fields Applied to Flow Data with Shock Waves. *Pages 161–177 of:* Palamidese, P. (ed), *Scientific Visualization: Advanced Software Techniques*. Ellis Horwood Limited.

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), *Proc. IEEE Visualization '94.*

Peikert, R., & Roth, M. 1999. The 'Parallel Vectors' Operator - A Vector Field Visualization Primitive. *Pages 263–270 of:* Ebert, D., Gross, M., & Hamann, B. (eds), *Proc. IEEE Visualization '99.*

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

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*, **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*. Second edn. Cambridge University Press.

Puig, A., Tost, D., & Navazo, I. 2000. Hybrid Model for Vascular tree Structures. *Pages 125–135 of*: de Leeuw, W., & van Liere, R. (eds), *Data Visualization 2000*. Springer Verlag.

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*. Springer Verlag.

Reinders, F., Spoelder, H.J.W., & Post, F.H. 1998. Experiments on the Accuracy of Feature Extraction. *Pages 49–58 of*: Bartz, D. (ed), *Visualization in Scientific Computing '98*. Springer Verlag.

Reinders, F., Post, F.H., & Spoelder, H.J.W. 1999a. Attribute-Based Feature Tracking. *Pages 63–72 of*: Gröller, E., Löffelmann, H., & Ribarsky, W. (eds), *Data Visualization '99*. Springer Verlag.

Reinders, F., Post, F.H., & Spoelder, H.J.W. 1999b. Visualization of Time-Dependent Data using Feature Tracking. *Pages 150–157 of*: Boasson, M., Kaandorp, J.A., Tonino, J.F.M., & Vosselman, M.G. (eds), *Proc. ASCI '99*. Advanced School for Computing and Imaging.

Reinders, F., Jacobson, M.E.D., & Post, F.H. 2000. Skeleton Graph Generation for Feature Shape Description. *Pages 73–82 of*: de Leeuw, W., & van Liere, R. (eds), *Data Visualization 2000*. Springer Verlag.

Reinders, F., Post, F.H., & Spoelder, H.J.W. 2001. Visualization of Time-Dependent Data using Feature Tracking and Event Detection. *The Visual Computer*. Accepted for publication.

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

Rossow, W.B., Genio, A.D. Del, Limaye, S.S., Travis, L.D., & Stone, P.H. 1980. Cloud Morphology and Motions from Pioneer Venus Images. *J. of Geophys. Res.*, **85**(13), 8107–8128.

Rossow, W.B., Genio, A.D. Del, & Eichler, T. 1990. Cloud-Tracked Winds from Pioneer Venus OCPP Images. *J. of Atmos. Sci.*, **47**(17), 2053–2082.

Roth, M., & Peikert, R. 1996. Flow Visualization for Turbomachinery Design. *Pages 381–384 of*: Yagel, R., & Nielson, G.M. (eds), *Proc. IEEE Visualization '96*.

Sadarjoen, I.A., & Post, F.H. 1999. Geometric Methods for Vortex Extraction. *Pages 53–62 of:* Gröller, E., Löffelmann, H., & Ribarsky, W. (eds), *Data Visualization '99*. Springer Verlag.

Sadarjoen, I.A., & Post, F.H. 2000. Detection, Quantification, and Tracking of Vorties using Streamline Geometry. *Computers & Graphics*, **24**, 333–341.

Salari, V., & Sethi, I.K. 1990. Feature Point Correspondence in the Presence of Occlusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **12**(1), 87–91.

Samtaney, R., Silver, D., Zabusky, N., & Cao, J. 1993 (Aug.). *Feature Tracking and Visualization*. CAIP Technical Report TR-17S. Rudgers University.

Samtaney, R., Silver, D., Zabusky, N., & Cao, J. 1994. Visualizing Features and Tracking Their Evolution. *IEEE Computer*, **27**(7), 20–27.

Scheuermann, G., Hagen, H., Kruger, H., Menzel, M., & Rockwood, A. 1997. Visualization of Higher Order Singularities in Vector Fields. *Pages 67–74 of:* Yagel, R., & Hagen, H. (eds), *Proc. IEEE Visualization '97*.

Schroeder, W.J., Zarge, J.A., & Lorensen, W.E. 1992 (July). Decimation of Triangle Meshes. *Pages 65–70 of:* Catmull, E.E. (ed), *Computer Graphics (Proc. SIGGRAPH '92)*, vol. 26.

Sethi, I.K., & Jain, R. 1987. Finding Trajectories of Feature Points in a Monocular Image Sequence. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **9**(1), 56–73.

Sethi, I.K., Patel, N.V., & Yoo, J.H. 1994. A General Approach for Token Correspondence. *Pattern Recognition*, **27**(12), 1775–1786.

Silver, D., & Wang, X. 1996. Volume Tracking. *Pages 157–164 of:* Yagel, R., & Nielson, G.M. (eds), *Proc. IEEE Visualization '96*.

Silver, D., & Wang, X. 1997. Tracking and Visualizing Turbulent 3D Features. *IEEE Trans. on Visualization and Computer Graphics*, **3**(2).

Silver, D., & Wang, X. 1998. Tracking Scalar Features in Unstructured DataSets. *Pages 79–86 of:* Ebert, D., Hagen, H., & Rushmeier, H. (eds), *Proc. IEEE Visualization '98*.

Silver, D., & Wang, X. 1999. Visualizing Evolving Scalar Phenomena. *Future Generation Computer Systems*, **15**(1), 99–108.

Silver, D., & Zabusky, N.J. 1993. Quantifying Visualizations for Reduced Modeling in Nonlinear Science: Extracting Structures from Data Sets. *J. of Visual Communication and Image Presentation*, **4**(1), 46–61.

Silver, D., Zabusky, N.J., Fernandez, V., Gao, M., & Samtaney, R. 1991. Ellipsoidal Quantification of Evolving Phenomena. *Pages 573–588 of:* Patrikalakis, N.M. (ed), *Scientific Visualization of Natural Phenomena.* Springer Verlag.

Smith, M.D., & Gierasch, P.J. 1996. Global-scale Winds at the Venus Cloud-top Inferred from Cloud Streak Orientation. *Icarus,* **123**, 313–323.

Sujudi, D., & Haimes, R. 1995. *Identification of Swirling Flow in 3-D Vector Fields.* Tech. rept. AIAA-95-1715. American Institute of Aeronautics and Astronautics, Inc., Washington, DC.

Toigo, A.D., Gierasch, P.J., , & Smith, M.D. 1994. High-resolution Cloud Feature Tracking on Venus by Galileo. *Icarus,* **109**, 318–336.

Vrolijk, B. 2000 (Feb.). *Feature Tracking met behulp van Greedy Exchange.* Task report. Delft University of Technology. Written in the dutch language.

van Walsum, T. 1995. *Selective Visualization on Curvilinear Grids.* Ph.D. thesis, Delft University of Technology, The Netherlands.

van Walsum, T., Post, F.H., Silver, D., & Post, F.J. 1996. Feature Extraction and Iconic Visualization. *IEEE Trans. on Visualization and Computer Graphics,* **2**(2), 111–119.

Weigle, C., & Banks, D.C. 1998. Extracting Iso-valued Features in 4-dimensional Scalar Fields. *Pages 103–110 of:* Lorensen, B., & Yagel, R. (eds), *Proc. IEEE Symp. on Volume Visualization '98.*

Wernecke, J. 1993. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor.* Addison-Wesley Publishing Company. Open Inventor Architecture Group.

Woo M. and Neider J. and Davis T. and Shreiner D. 1999. *OpenGL Programming Guide: The Official Guide to Learning OpenGL.* Third edn. Addison-Wesley Publishing Company. Open Inventor Architecture Group.

Xia, Y. 1989. Skeletonization via Realization of the Fire Front's Propagation and Extinction in Digital Binary Shapes. *IEEE Trans. on Pattern Recognition and Machine Intelligence,* **11**(10), 1076–1086.

Xu, X.W. 1996. Image Feature Analysis for Computer-aided Diagnosis: Detection of Right and Left Hemidiaphragm Edges and Delineation of Lung Field in Chest Radiograms. *Medical Physics,* **23**(9), 1613–1624.

Summary

This thesis focuses on feature-based visualization of time-dependent data sets with the goal of gaining insight in the evolving phenomena. This is achieved by the following four steps:

1. **Feature Extraction.** Extract the features in the data for each time-step (frame) and describe the characteristics of the features by calculating a number of attribute sets.

2. **Feature Tracking.** Track features by solving the frame-to-frame correspondence problem between features, using the attributes calculated in the first step.

3. **Event Detection.** Detect certain events in life-cycle of a feature, such as unusual changes, particular stages in the evolution of a feature, or specific interactions between features.

4. **Visualization.** Visualize the evolution of features interactively in a player, show the relations between features in successive frames, and highlight particular events.

Chapter 2 contains a survey of related work in the field of feature-based visualization. It illustrates the starting point of the research and serves as a foundation for the rest of the thesis.

The first part of this thesis focuses on the extraction and quantification of features.

Chapter 3 introduces a concept for the representation of feature data. The feature data representation stores the feature attributes and provides a number of functions and operations for the manipulation and comparison of features. This representation signifies a huge data reduction and is ideal for the analysis of time-dependent phenomena.

Chapter 4 addresses accuracy issues. The accuracy and stability of attribute calculation methods is verified, so that the feature attributes can be trusted and used for further analysis. The accuracy, and stability with respect to noise is verified using an experimental approach with synthetic data. In this way, we verified the accuracy of two attribute calculation methods: the volume integrals, and the ellipsoid fitting method.

Chapter 5 provides a new attribute calculation method describing the geometric shape of features by a skeleton description. Using the skeleton and the distance transform of a feature, a skeleton graph representation is calculated that describes the shape of the feature with a controlled precision. The skeleton graph can then be used to approximate the surface of the original shape.

The second part of this thesis focuses on the analysis of feature evolution.

Chapter 6 discusses the tracking of features in time using the feature data representation only, i.e. without using the original grid data. Each type of attribute set is associated to a number of correspondence functions that can be used to compare two features. Using these correspondence functions, the features are tracked based on a prediction-verification scheme which leads to smoothly evolving, continuous paths describing the evolution of objects.

Chapter 7 shows the detection of significant events in the evolution of objects. The following events can be detected: continuation, birth/death, entry/exit, split/merge, and topological loop/junction events. Again, the events are recognized using the feature data representation only. Each attribute set is associated to a number of event functions that test the criteria for a particular event.

Sections 6.5 and 7.7 describes the visualization of feature evolution by means of a linked combination of two viewers: the feature viewer and the graph viewer. The feature viewer shows the features in 3D space by iconic representations. The graph viewer shows the event graph which is the result of the feature tracking and event detection stage. With these viewers the user can interactively investigate the tracking results, and analyze the evolution of the features.

Samenvatting

Dit proefschrift richt zich op feature-gebaseerde visualisatie van tijdsafhankelijke datasets met het doel om inzicht te krijgen in evoluerende verschijnselen. Dit wordt bereikt met de volgende vier stappen:

1. **Feature Extraction**. Detecteer de features in de data van elke tijdstap (frame) en beschrijf de karakteristieken van de features door de berekening van een aantal sets van attributen.

2. **Feature Tracking**. Volg de features in tijd door het frame-to-frame correspondentie probleem tussen features op te lossen. Hierbij wordt enkel gebruik gemaakt van de attributen die in de eerste stap zijn berekend.

3. **Event Detection**. Detecteer bepaalde gebeurtenissen (events) in de levensloop van features, zoals ongebruikelijke veranderingen, bepaalde fasen in de ontwikkeling van een feature, of specifieke interacties tussen features.

4. **Visualization**. Visualiseer de ontwikkeling van features interactief in een speler, laat de relaties tussen features in opeenvolgende frames zien en leg de nadruk op bepaalde events.

Hoofdstuk 2 geeft een overzicht van verwant werk in het gebied van feature-gebaseerde visualisatie. Dit hoofdstuk illustreert de uitgangspositie van het onderzoek en vormt de basis voor de rest van het proefschrift.

Het eerste gedeelte van dit proefschrift richt zich op de extractie en quantificatie van features.

Hoofdstuk 3 introduceert een concept voor de representatie van feature data. De feature data representatie maakt het mogelijk om de feature attributen op te slaan en geeft een aantal functies en operaties waarmee features kunnen worden gemanipuleerd en vergeleken. Bovendien houdt de representatie een grote data reductie in, waardoor deze ideaal is voor het analyseren van tijdsafhankelijke fenomenen.

Hoofdstuk 4 behandelt nauwkeurigheidszaken. De nauwkeurigheid en stabiliteit van attribuut bepaling is geverifieerd zodat de attributen kunnen worden vertrouwd bij verdere analyses. De nauwkeurigheid en stabiliteit in aanwezigheid van ruis is bepaald met behulp van een experimentele aanpak die gebruik maakt van synthetische data. Op deze wijze is de nauwkeurigheid van twee attribuut berekenings methoden, de volume integralen en de ellipsoide fitting methode, geverifieerd.

Hoofdstuk 5 behandelt een nieuwe manier van attribuut berekening die de geometrische vorm van features bepaalt met behulp van skelet informatie. Een skelet graaf representatie, die de vorm van een feature beschrijft met een gecontroleerde precisie, kan worden bepaald met behulp van het skelet en de distance transform van een feature. Deze graaf kan vervolgens gebruikt worden om het oppervlak van de oorspronkelijke vorm te benaderen.

Het tweede gedeelte van het proefschrift richt zich op de analyse van feature evolutie in tijd.

Hoofdstuk 6 beschrijft het volgen van de features in de tijd, waarbij alleen gebruik gemaakt wordt van de feature data representatie, i.e. de originele rooster data wordt niet gebruikt. Met elk type attribuut set wordt een aantal correspondentie functies verbonden die gebruikt kunnen worden om twee features te vergelijken. Met behulp van deze correspondentie functies kunnen de features worden getrackt gebaseerd op een predictie-verificatie schema die resulteert in glad verlopende, continuerende paden die de evolutie van objecten beschrijven.

Hoofdstuk 7 laat de detectie zien van betekenisvolle events in de evolutie van de objecten. De volgende typen events kunnen worden herkend: voortzetting, geboorte/dood, inkomen/uitgaan, splitsen/samenvloeien en topologische lus/knooppunt events. Deze events kunnen wederom herkend worden met enkel de hulp van de feature data representatie. Elke attribuut set is verbonden met een aantal event functies die de criteria voor een specifiek event testen.

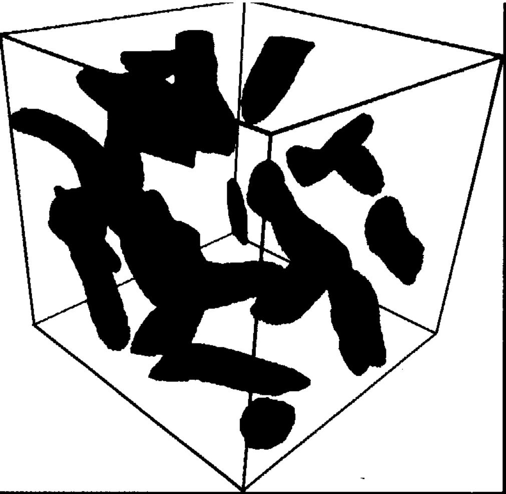
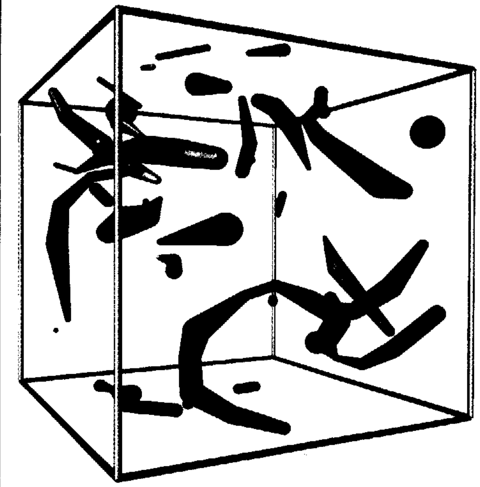
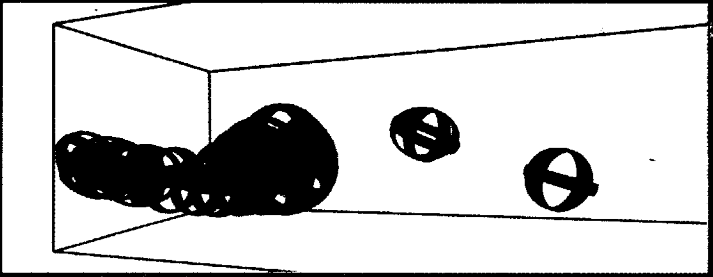
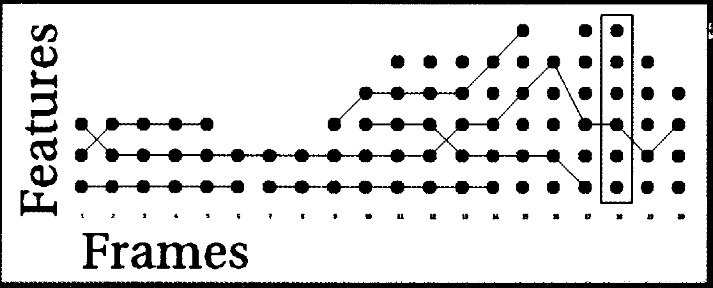
De paragrafen 6.5 en 7.7 beschrijven de visualisatie van de feature evoluties met behulp van een combinatie van twee gerelateerde viewers: de feature viewer en de graaf viewer. De feature viewer laat de features zien in de 3D ruimte met behulp van iconische representaties. De graaf viewer laat de event graaf zien die resulteert bij het feature tracking en event detectie proces. De gebruiker kan met deze twee viewers de resultaten van de tracking interactief bekijken en onderzoeken; zo kan de evolutie van de features worden geanalyseerd.

Curriculum Vitæ

Freek Reinders was born on the 24th of April 1970 in Emmeloord, Noordoostpolder, The Netherlands. In 1989, he received his VWO diploma at the Christelijke Scholen Gemeenschap in Emmeloord. He received his engineer's degree in Applied Physics in 1995 for a project carried out at the Biomagnetic Centre at the University of Twente in Enschede. The subject of his graduation work was the simulation of brain signals (EEG and MEG) using the finite-element method.

In March 1996, he started his PhD research project in the Computer Graphics & CAD/CAM group of Delft University of Technology. The main topic of the research project involved scientific visualization, in particular the feature-based visualization of time-dependent data sets.

Since February 2001, he works at Medis, Medical Imaging Systems, in Leiden. The focus of his work is directed to the 3D visualization of MR Angiography.





Features
Frames
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20