Selective visualization on curvilinear grids
About the front cover

An iconic visualization of flow on a curvilinear grid. It shows streamlines in a blunt fin geometry, and some flow probes along one of these streamlines. Flow probes were developed by Wim de Leeuw [45].

About the back cover

*Upper image:* two streamlines in a backward facing step geometry. The width of the streamlines is inversely proportional to the square of the velocity magnitude, the colour of the streamlines is determined by the pressure. The streamlines pass through a region of high normalized helicity density, shown by the ellipsoid icons. Data courtesy Numerical Mathematics Department, Delft University of Technology.

*Middle image:* regions of high reaction speed in an atmospheric flow with a chemical reaction. Gray ellipsoid icons visualize regions, large arrows display average velocity over these regions, and red/yellow ellipsoid icons display velocity distribution over these regions. Data courtesy Laboratory for Aero- and Hydrodynamics, Delft University of Technology.

*Lower image:* visualization of a scalar variable (normalized dot product of turbulent mass flux and concentration gradient) inside a plume flow. Data courtesy Laboratory for Aero- and Hydrodynamics, Delft University of Technology.
Selective visualization on curvilinear grids

Selectieve visualisatie op kromlijnige roosters

PROEFSCHRIFT
ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus Prof.ir. K.F. Wakker
in het openbaar te verdedigen ten overstaan van een commissie,
door het College van Dekanen aangewezen,
on maandag 18 december 1995 om 13.30 uur
door

Theodores VAN WALSEUM
informatica ingenieur
geboren te Krimpen aan de IJssel
Dit proefschrift is goedgekeurd door de promotoren:
Prof.dr.ir. F.W. Jansen
Prof.dr.ir. P. Wesseling

Toegevoegd promotor:
Ir. F.H. Post

Samenstelling promotiecommissie:
Rector Magnificus, voorzitter
Prof.dr.ir. F.W. Jansen, Technische Universiteit Delft, promotor
Prof.dr.ir. P. Wesseling, Technische Universiteit Delft, promotor
Ir. F.H. Post, Technische Universiteit Delft, toegevoegd promotor
Prof.dr.ir. F.C.A. Groen, Universiteit van Amsterdam
Prof.dr. L. Hesselink, Stanford University
Prof.dr.ir. F.T.M. Nieuwstadt, Technische Universiteit Delft
Prof.dr.ir. M.A. Viergever, Rijksuniverseit Utrecht

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Walsum, Theodores van
Selective visualization on curvilinear grids / Theodores van Walsum. - [S.l. : s.n.]. Ill.
Thesis Technische Universiteit Delft. - With ref. - With summary in Dutch.
ISBN 90-9008871-7
NUGI 855
Subject heading: visualization / curvilinear grids / data selection.
Preface

The research described in this thesis was carried out at the Computer Graphics Group of the Faculty of Technical Mathematics and Informatics of Delft University of Technology. It is the second PhD project in a series of several projects on visualization. Visualization research at this group has mainly been focused on visualization of 3D fluid flow, partly because of relations with groups that work on simulation and modelling of fluid flow: the Numerical Mathematics Group and the Laboratory for Aero- and Hydrodynamics at Delft University of Technology, and Delft Hydraulics.

Each PhD project has its own focus in the field of visualization of 3D fluid flow. Visualization of a special flow phenomenon, turbulence, was the subject of a recently finished project by Andrea Hin [30]. In another project (Wim de Leeuw), focus is on interaction and rendering techniques for flow visualization. In my project, the focus is on techniques for curvilinear grids, and on selective visualization techniques and in that way is a complement to the other projects.

Several people have contributed to the work reported in this thesis, in many different ways. Frits Post, my main supervisor, was the one who inspired me. The many discussions with him stimulated me, and were deciding for the directions I chose. I want to thank him for his support and advice, and for giving me the opportunity to work on the things I wanted. Guus Segal, my other supervisor, kept me realising that techniques should be applicable in practice, preferably in the practice of numerical simulations.

I want to thank my promotors, professors Jansen and Wesseling, for the opportunity to do a PhD project at their department, and for their support and advice during this project.

Two Master’s students contributed to this project. Ari Sadarjoen worked on particle tracing algorithms and differencing techniques for Jacobian calculations. His work convinced us that particle tracing should be done in physical space. Frank Post worked on iconic visualization, and implemented the techniques described in Chapter 5. His enthusiasm and zeal made cooperation with him a privilege.

My roommates Ari Sadarjoen and Andrea Hin, and Erik Reinhard, Klaas-Jan de Kraker, Maurice Dohmen, Wim de Leeuw and Winfried van Holland, the other inhabitants of the AIO Lab, formed a pleasant work environment, and created an
atmosphere which I enjoyed very much.

Ivo Bouwmans, John Meeder and Jaap den Toonder, of the Laboratory for Aero- and Hydrodynamics of Delft University of Technology, provided me with interesting datasets, and interesting research questions, which is at least as important as an interesting dataset. Without their cooperation, there would have been no chapter with case studies.

The technical staff, Aadjan van der Helm, Kees Seebregts, Peter Kailuhu and Piter Jonker, kept the HP's and SGI's running, and provided the necessary technical support, that is indispensable for the work of a computer scientist.

Finally, I want to thank Paula, for encouraging me and being patient with me. Especially the last half year, when I had to finish my thesis in the evening hours, her support was invaluable.

Theo van Walsum
Contents

Preface v

Contents vii

1 Introduction 1
  1.1 Visualization in scientific computing 1
  1.2 Objectives 2
  1.3 Overview of this thesis 3

2 Overview of visualization techniques 5
  2.1 Visualization pipeline 5
  2.2 Data generation 7
  2.3 Data preparation 10
  2.4 Visualization mapping 11
    2.4.1 Scalar data mappings 12
    2.4.2 Vector data mappings 14
    2.4.3 Tensor data mappings 20
  2.5 Rendering and Display 21
    2.5.1 Line and polygon rendering 22
    2.5.2 Volume rendering 22
    2.5.3 Hybrid rendering 24
    2.5.4 Animations 25
  2.6 Research objectives 26

3 Visualization techniques on curvilinear grids 29
  3.1 Introduction 29
  3.2 Grid transformation 32
  3.3 Point location in $P$ 36
    3.3.1 Stencil Walk 37
    3.3.2 Bilinear cell boundaries 39
    3.3.3 Tetrahedrization 43
    3.3.4 Comparison 44
**CONTENTS**

3.4 Interpolation in $\mathcal{P}$ .................................................. 52
  3.4.1 Interpolation techniques ........................................... 52
  3.4.2 Comparison .......................................................... 53
3.5 Example ................................................................. 60
3.6 Conclusions ............................................................ 61

4 Selective visualization .................................................. 63
  4.1 Selective visualization model ........................................ 64
  4.2 Selection creation .................................................... 66
    4.2.1 Selection expressions ........................................... 67
    4.2.2 Representation of selection ................................... 69
    4.2.3 Evaluation of expressions ..................................... 72
  4.3 Selection processing ................................................ 76
  4.4 Selective visualization mapping ................................... 79
    4.4.1 Direct visualization of selections ............................ 80
    4.4.2 Clipping with selections ....................................... 80
    4.4.3 Parameter extraction using selections ....................... 85
  4.5 Example: candidate cells for critical points .................... 86
  4.6 Conclusions .......................................................... 91

5 Iconic visualization .................................................... 93
  5.1 Iconic visualization model ......................................... 94
  5.2 Attributes ........................................................... 96
    5.2.1 Volume integrals for attribute calculation ................ 96
    5.2.2 Other techniques for attribute calculation ............... 99
    5.2.3 Attribute set transformation .................................. 100
  5.3 Iconic visualization mapping ....................................... 101
    5.3.1 Icons .......................................................... 101
    5.3.2 Icon modelling language ..................................... 102
    5.3.3 Example icons ................................................ 102
  5.4 Examples of iconic visualization .................................. 105
  5.5 Conclusions .......................................................... 108

6 Case studies ............................................................ 111
  6.1 Backward facing step ............................................... 111
  6.2 Turbulent pipe flow ................................................ 114
  6.3 Atmospheric flow with a chemical reaction ....................... 119
  6.4 Atmospheric flow with plume ...................................... 122
  6.5 Conclusions .......................................................... 127

7 Conclusions ............................................................. 129
  7.1 Techniques for curvilinear grids .................................. 129
  7.2 Selective visualization ............................................. 130
  7.3 Iconic visualization ............................................... 131

viii
CONTENTS

7.4 Future research ........................................... 131

References ................................................................ 133

A AVS Modules ................................................... 139
A.1 Input modules ................................................. 139
A.2 Filter modules ................................................. 140
A.3 Mapper modules .............................................. 142
A.4 Output modules .............................................. 143

B Syntax of input for cbs module ......................... 145
B.1 Expressions ................................................... 145
B.2 Commands .................................................... 146
B.3 Queries ......................................................... 146
B.4 Arithmetic .................................................... 147

C C-macros for icon modelling ......................... 151
C.1 Attribute binding ........................................... 151
C.2 2D contour curve definition ......................... 152
C.3 From 2D contour to 3D geometry .................. 153
C.4 3D transformations ....................................... 153
C.5 Icon administration ...................................... 154
C.6 Code for some example icons ....................... 155

D Datasets ........................................................ 157
D.1 Blunt fin ....................................................... 157
D.2 Hydrogen ...................................................... 157
D.3 Backward facing step .................................... 157
D.4 Turbulent pipe flow ...................................... 158
D.5 Flow with chemical reaction ...................... 158
D.6 Plume flow ................................................... 158

E Colour plates .................................................. 159

Summary ................................................................ 167

Samenvatting ................................................................ 169

Curriculum Vitae .................................................. 171
Chapter 1

Introduction

The computational power of computers is still increasing. As a result, computer simulations are still growing: in size, dimensionality and complexity. These simulations result in data sets, that consequently grow in size, dimensionality and complexity. Therefore, there is a growing need for tools to analyse this data; tools for data analysis are indispensable for modern natural sciences.

Traditionally, data has been analysed with statistics and 2D graphs. Unfortunately, these can meet the requirements no longer. Statistics cannot reveal all important aspects of large datasets, whereas simple 2D graphs cannot represent all data of a large 3D dataset. Additional tools are needed, and visualization is one of the approaches that gives opportunities to analyse large scientific datasets.

1.1 Visualization in scientific computing

Visualization is a process that transforms data into visible objects, in order to let the human visual system analyse the data. 2D graphs are an example of visualization of data, but visualization encompasses many other techniques that enable visual analysis of large datasets.

In the last decade, visualization has become an important research topic in computer science. The needs for sophisticated tools for visual data analyses were recognized in the NSF report that appeared in 1987 [54]. In this report, visualization is described as follows:

Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science. ([54], page 3)
CHAPTER 1. INTRODUCTION

The transformation of symbolic (or numerical) data to geometric data is central to visualization. If an appropriate transformation is chosen, the scientist really sees the data, and consequently can explore and analyse the data visually. As our visual system is very well adapted to analysing visual representations of data, visualization can contribute to scientific and engineering research.

In the NSF report, three functions of visualization are described. The first, and most important, is the exploration of data. Scientists need to see their data in order to interpret the data.

A second function is communication. Visual information can aid the sharing of results with colleagues all around the world. The well-known saying that an image may say more than a thousand words is certainly applicable here.

A third function is steering of large simulations. Scientists want to interpret what is happening to a model during computations. Instead of analysing data after the simulation is finished, they want to change parameters, and interact with their model during the simulation.

1.2 Objectives

In our work, we will mainly focus on the first function of visualization: data exploration. This does not mean that we do not pay any attention to the other functions: tools for data exploration might give images that are very suitable for communication, and the same tools might be used for steering computations. It is, however, not our main purpose to develop techniques that generate images for communication or steering, we want to provide the scientist with techniques to explore the data.

Furthermore, we restrict ourselves to simulation data. This means that we do not work on techniques for experimental data, or techniques for visualization of experiments, such as wind tunnel devices. Note that we also use visualization in this restricted sense: we use it as a synonym for visualization in scientific computing, while traditionally visualization applies to techniques to visualize (fluid flow) experiments.

A further restriction is that we assume that the data is represented on a single block Cartesian or curvilinear grid. This means that we will not provide techniques for multi-block grids, or unstructured grids. Curvilinear grids are frequently applied in Computational Fluid Dynamics (CFD), which is our main application area.

Within these restrictions, the objectives of our research project are twofold:

- to develop techniques that can deal with data defined on 3D curvilinear grids
- to develop techniques that allow for a reduction of data and extraction of information in the visualization process
We will discuss these objectives in the context of the current state of the art in visualization at the end of Chapter 2.

1.3 Overview of this thesis

The remainder of this thesis is organized as follows. We will first give an overview of visualization techniques for 3D scalar and vector datasets (Chapter 2). This overview is followed by a discussion of current problems in visualization, on which the definition of our objectives is based.

Then we continue with a chapter on visualization techniques for curvilinear grids (Chapter 3). Two closely related operations, point location and interpolation, are discussed in this chapter. We describe and compare techniques that can be used in 3D curvilinear grids.

In Chapter 4 and 5 we deal with data reduction techniques for visualization of large datasets. Chapter 4, on selective visualization, describes a technique to select parts of a dataset, and the use of these selected data in the visualization mapping. Chapter 5, on iconic visualization, describes an extra step that can be added to selective visualization, in order to visualize selected data with iconic objects. In Chapter 6, we apply these techniques described in four cases.

Finally, Chapter 7 contains the conclusions of the work presented in this thesis, and directions for future research.
Chapter 2

Overview of visualization techniques

In this chapter, we give an overview of scientific visualization techniques. This overview serves two functions: it introduces the reader to the state of the art in visualization, thereby explaining the basic visualization concepts, and it is the basis for the definition of our research objectives. The overview is limited to scientific data datasets on structured grids, and we will focus on visualization techniques for fluid flow computations. It is partially based on an earlier given overview of fluid flow visualization techniques [62].

As stated in the introduction, we do not address "experimental" fluid flow visualization techniques, i.e. the techniques that are used in physical experiments, such as wind tunnels. Merzkirch [55] and Yang [93] give overviews of various kinds of such flow visualization techniques.

The visualization process, which transforms numerical data into images, can be modelled as a pipeline. We use this pipeline model, the visualization pipeline, as a basis for the overview. This model is described in the next section. The stages of the pipeline are described in the sections following the pipeline description. In the last section of this chapter, we summarize and discuss some problems related to current visualization techniques. Based on these problems, our research objectives are discussed in more detail.

2.1 Visualization pipeline

The visualization process can be described by a pipeline model, called the visualization pipeline [25, 26]. This pipeline consists of a sequence of transformation operations that are applied to the original data. In Figure 2.1 the visualization pipeline is shown.

The pipeline model divides the visualization process in several stages with
Figure 2.1: Visualization pipeline, boxes contain transformations.
different functions. In practice, the visualization process does not have to contain all stages identified in Figure 2.1. If the data produced by the simulation can be used directly by the visualization mapping, data preparation can be skipped. Also, it might be argued that the first stage does not belong to the visualization process. If one considers visualization as a post processing operation, data generation is not a part of the visualization process. However, if visualization and simulation are closely interconnected, as is the case in steering environments [51], data generation is the first stage of the visualization process.

The subdivision of the visualization process in a number of stages with different functions is used by commercial visualization programs such as AVS, IRIS Explorer and Khoros [1, 75, 15]. These applications consist of a set of modules that implement a specific function, and that can be linked together in a pipeline (network in AVS terms). The visualization program sends the data through the network, and executes the modules to process the data. Figure 2.2 shows a pipeline for AVS, that generates an isosurface for velocity magnitude. The top module, read field, reads the data from disk (interface with data generation), the second module, vector mag, calculates the velocity magnitude from the velocity field (data preparation), the third module, isosurface, generates an isosurface for a given isovalue (visualization mapping), and the fourth module, geometry viewer, renders the isosurface and displays the image on the screen (rendering and display).

2.2 Data generation

In Figure 2.1, data generation is shown as the first stage of the visualization process. Whether included in the visualization process or not, the consequences
of the aspects of data generation relevant for the visualization process need to be discussed. Three aspects of data generation are relevant for the remainder of the visualization pipeline:

- the origins of the data,
- the grid type of the data,
- the size of the data set.

Data may originate from two types of sources: experimental measurements or numerical simulations. The characteristics of experimentally measured data are mainly determined by the measuring equipment. The equipment will introduce noise in the data, therefore filtering techniques might be necessary in the data preparation stage. The measuring equipment also determines the positions of the data samples, and the type of measured data. Not all quantities can be measured in experiments, and measurements often can only be performed in one or two dimensions. Therefore, experimental measured data will generally need a number of filtering steps before it can be fed in the following stage of the pipeline.

Data that originate from numerical simulations have fewer restrictions than measured data: there is no noise from the measuring equipment, the domain is specified by the grid, and with current grid generation techniques a wide variety of domains can be used. Furthermore, there is no limit to the quantities that can be calculated. Thus it follows that numerical simulations are likely to cause the largest problems for visualization, as there is virtually no restriction on the dimensionality and size of the dataset and the number and type of quantities calculated.

Numerical simulations are performed on a grid: the quantities are calculated at specific positions (nodes) in the domain. There are several types of grids used in numerical simulations, see Figure 2.3. The simplest grid type is a Cartesian grid: a grid with a regular topology (structured grid), with the nodes in a evenly-spaced rectangular pattern (regular geometry). Curvilinear grids also have a regular topology, but they do not have a regular geometry. Unstructured grids do not have a regular topology. Cartesian and curvilinear grids are used for finite volume and finite difference techniques, unstructured grids are used for finite element techniques.

Some simulation techniques use staggered grids, a special form of the above mentioned grid types. In these grids, not all quantities are calculated at the same positions, e.g. some CFD simulations calculate pressure at the cell centres, and velocity components at cell boundaries.

For some 3D scalar datasets on Cartesian grids, the data values are specified in the centres of the cells, or the data values are assumed to represent average values for a cell. Such cells are called voxels.

The size of the dataset depends on the grid size and the dimensionality of the grid. Simulations may vary from 2D stationary simulations of only a few thousand nodes, to instationary 4D (three spatial dimensions and a time dimension)
Figure 2.3: Grid types; a) Cartesian grid; b) unstructured grid; c) curvilinear grid.
simulations of billions of nodes. File sizes consequently vary from a few hundred Kbytes to several Gbytes. This is not just a quantitative change: such a large difference implies a qualitative change as well, both for simulation and visualization techniques.

2.3 Data preparation

Data preparation is the first step of the visualization process after data is generated. The purpose of this stage is to transform the original data to a form that can be used by the next stage in the visualization process. The input of this stage consists of raw data from the generation stage and the output is data suitable for visualization. Operations in this stage involve smoothing, resampling and interpolation, selection and calculation of derived quantities.

*Smoothing* techniques are necessary if the data is noisy; it is used to remove noise introduced in data generation. It can also be used if the visualization mapping needs a smooth data field. E.g. if a very noisy 3D scalar dataset is visualized with an isosurface technique, the visualization may reveal no information other than that the dataset is very noisy. A solution is to filter out the high frequencies, and apply the same visualization technique to the filtered data. The resulting visualization will give more information on the data. Of course, another solution is to use a visualization technique that contain an implicit smoothing effect, such as a direct volume rendering technique.

*Resampling* techniques are necessary if the dataset is represented on a grid that can not be used in the visualization mapping. Some visualization techniques (such as many 3D direct volume rendering techniques) are applicable only to datasets on simple Cartesian grids. If the data are represented on other, more complex grids, the dataset has to be mapped on a simpler grid. This involves interpolation of data. However, complex grids are used because they have advantages such as boundary conformance and local variation of grid spacing. Therefore, resampling to a simpler grid will almost always lead to a significant loss of information and/or a significant increase in size of the dataset.

*Selection* techniques are necessary if there is too much data to be visualized directly. Some visualization techniques, such as colour contours, only operate on 2D datasets. If a dataset is 3D, a 2D slice may be selected, to which the visualization technique is applied. This reduction of a 3D dataset to a 2D slice is the best-known, and probably the most widely used, selection technique. A global view of the dataset can be obtained by moving the cutting plane through the dataset.

Selection techniques are related to resampling techniques: if a 2D slice of a 3D dataset does not coincide with a layer of the 3D grid, data has to be interpolated at new grid positions on the 2D slice. However, as the dimensionality of the selection grid is lower than the dimensionality of the original grid, the accuracy and data size problems are less severe than in global resampling.
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Data derivation techniques are necessary if the data to be used in the visualization mapping are not directly available in the dataset, but can be calculated from data in the dataset. An example is given in Figure 2.2, where vector magnitude is calculated. Such an operation, where a vector quantity is transformed to a scalar quantity, is called a contraction. Other data derivation examples for CFD datasets are:

\[ \mathbf{v} = \rho \mathbf{m} \]  
\[ \omega = \nabla \times \mathbf{v} \]  
\[ h = \mathbf{v} \cdot \omega \]

(2.1) \hspace{1cm} (2.2) \hspace{1cm} (2.3)

where \( \mathbf{m} \) is momentum, \( \rho \) is density, \( \mathbf{v} \) is velocity, \( \omega \) is vorticity, \( h \) is helicity and \( \nabla \) the gradient operator. These operations apply for flow field visualization, for other application areas, similar transformations can be applied.

2.4 Visualization mapping

Visualization mapping is the stage where physical data is translated to suitable visual primitives and attributes. This is the central part of the process; it involves the "design" of the visualization: to determine what we want to see, and how to visualize it. Physical quantities are cast into a visual domain of shapes, light, colour and other optical properties.

In the visualization mapping stage, it is determined how information is retrieved from the data: the mapping either lets the scientist extract the information by visual interpretation a large amount of visually presented data, or the mapping presents the information directly, after application of some technique to extract the information from the data. Most mappings are based on the first approach: all data is mapped to visual primitives that are projected onto an image, and the scientist deduces information by visually processing the images. An example is the mapping of all vectors of a vector field to arrows, and displaying all arrows. For 2D datasets this approach works reasonably well, as the human visual system is quite powerful in interpreting visual data. For 3D datasets, however, the amount of data often is too large to be projected directly. An example of the other approach is vector field topology. The information, a vector field's topology, is extracted from the data prior to visualization. The resulting topology can be presented directly to the scientist. These automated information analysis techniques are rare at present for scientific data.

The actual mapping is carried out by computing from the data derived quantities that are suitable for rendering. The transformation in this stage is from numerical data to geometric objects (or objects that can be rendered). Haber and McNabb [25] call these objects AVO's (Abstract Visualization Objects); in AVS they are called "geometries". These terms are slightly misleading, as the data is not always transformed to an object or geometry, as is the case with direct
volume rendering. We therefore call the output of this stage with a more general term visual primitives.

Data can be translated to visual primitives in many different ways. Indeed, much effort in scientific visualization research has been invested in developing new mappings. For each data type, several mappings are available. Below we discuss mappings for the main data types: scalar, vector, and tensor.

2.4.1 Scalar data mappings

2D scalar data are relatively easy to visualize. For 2D scalar datasets, some standard mappings are common: height fields, colour coding and contour lines, see Figure 2.4. With height fields, the scalar value is mapped to an offset with respect to a base plane. A surface (or grid) is drawn through the resulting positions, displaying the 2D scalar field as a landscape with hills and valleys. With colour coding, the scalar value is mapped to a colour scale. A colour is assigned to each scalar value, yielding a coloured plane where each colour denotes a specific value (or range of values) of the scalar. Contour lines are lines on a surface where the scalar value is constant.

Of course, these techniques can be combined: height fields can be colour coded, colour coding can be combined with contour lines, etc. These techniques can be used simultaneously to get multiple cues for interpreting one scalar variable, or to visualize two or three scalar variables on the same domain.

For 3D scalar datasets, there are techniques analogous to the techniques for 2D

![Figure 2.4: Mappings for 2D scalar datasets (applied on a slice of a 3D dataset): a) height fields; b) colour coding.](image-url)
Figure 2.5: Isosurface visualization.

scalar datasets. Colour coding for 3D datasets can be done by assigning a colour and opacity value to each scalar value. The opacity value determines the degree of transparency for an element of the scalar dataset. This mapping is often used in medical imaging: a colour and opacity value are assigned to each data value in an MRI dataset, corresponding to the properties of the tissue. Additionally a normal vector, based on the gradients of the scalar data field, can be used for shading a boundary surface between two tissues. Another application of this mapping is the visualization of concentrations as densities. With this mapping, a more or less realistic visualization can be achieved. These mappings can be rendered with direct volume rendering techniques (see Section 2.5), which are capable of rendering semi transparent volumes, and treat the scalar dataset as a continuous field.

The 3D analogy of 2D contours are isosurfaces (see Figure 2.5): surfaces where the scalar variable has a constant value. In contrast to 2D scalar datasets, where the number of contours displayed can be arbitrarily high, 3D datasets only permit to show a few isosurfaces. Isosurfaces are concentric, separate isosurfaces cannot be perceived as distinct anymore if it is attempted to visualize more than three or four isosurfaces. An algorithm for the construction of isosurfaces is described in [48], extensions to this algorithm are described in [59].

Alternatively, each scalar value at a grid position can be mapped to a coloured object, such as a sphere or a box, with a transparency. Colour, transparency and size of the object are parameters that can visualize a scalar variable (see e.g. [58]).
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Figure 2.6: Examples of multiple slices to visualize a 3D scalar data set

The above mappings for 3D scalar datasets share a problem that is common to most mappings for 3D datasets: it is difficult to create a mapping that visualizes the whole dataset in a single 2D image. There will often be objects occluding other objects, and thus hiding possibly relevant data.

There are other mappings for 3D scalar datasets that deal with this problem in a different way: instead of visualizing the whole dataset, a selected part of the dataset is visualized. Common approaches are to select one or more slices from a 3D dataset, and use an appropriate 2D mapping on these slices. This only visualizes the data at the slice, but the visualization of the data on the slice will be clearer than a visualization of the whole dataset. To get a global view of the data, a series of slices, or an orthogonal set of slices can be used (see Figure 2.6). Slices moving through the dataset also provide a view of the whole 3D dataset, although the scientist has to mentally reconstruct the global view, instead of seeing it directly.

2.4.2 Vector data mappings

There are two formalisms for numerically generating fluid flow data: Eulerian, computing quantities at fixed locations, and Lagrangian, computing quantities along a particle's path. Likewise, mappings for vector data can be divided into two styles: Eulerian mappings visualize vector data at fixed locations, whereas Lagrangian mappings visualize vector data along the trajectory of a particle moving in the vector field. A third style might be called indirect visualization: not
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

the vector field itself, but derived quantities, such as vector magnitude or helicity or the topology of a vector field are visualized.

Some techniques for visualization of vector fields are analogues of, or strongly related to, techniques used in experimental flow visualization. Experimental flow visualization techniques, based on addition of foreign material to the flow result in particle paths, streak lines and time lines. These concepts are described by Merzkirch [55]:

- a path line is the path of a particle in the fluid. Imagine a light-emitting particle in flow. A path line is obtained when a photographic plate is exposed for several seconds;

- a streak line arises when dye is injected in the flow from a fixed position. Injecting the dye for a period of time gives a line of dye in the fluid, from which the fluid flow can be seen;

- a time line is a line that, once released in the fluid, is moved and transformed by the fluid flow. The motion and formation of the line, which is often released perpendicular to the flow, shows the fluid flow.

In addition, streamlines are lines that are tangent to the flow everywhere. Streamlines for one time step in an instationary flow are called instantaneous streamlines. For stationary flows, streamlines, particle paths and streak lines coincide.

These concepts from experimental flow visualization also occur in computer generated visualizations of numerical vector field simulations. In the following common mappings for 2D and 3D vector fields are discussed, followed by some remarks on simulations of optical flow visualization techniques. We end with a description of flow field topology visualization and feature-based approaches.

Common mappings for 2D vector fields are arrow plots, streamlines and particle traces. An arrow plot is the Eulerian style mapping for a vector field (see Figure 2.7): a vector quantity is shown at a number of fixed locations at the same instant with an arrow-shaped icon [44]. The shape of the arrow can

Figure 2.7: 2D arrow plot of a vector field
be tuned to give the best visual result. The effect of various arrow shapes for visualizing vector data is discussed by Kroos [44]. Generally, lines, triangles or arrows consisting of a cylinder and cone are used. These icons scale with the vector magnitude, and point in the direction of the vector.

Particle traces, streak lines, time lines and streamlines are Lagrangian style mappings for vector datasets (see Figure 2.8). The path of an infinitesimal massless particle in the flow is computed and visualized. The particle motion can be shown using animation techniques. Particle attributes, such as shape, size and colour, can be used for mapping other quantities. E.g. a scalar quantity can be used to assign a colour to a particle. A particle trace or streak line can also be displayed as a continuous line, analogous to stream lines. In that case, vector magnitude information is not shown. However, line attributes such as colour, might be used to map vector magnitude information. Also, for 2D incompressible fluid flows, velocity magnitude is inversely proportional to the distance between adjacent streamlines, so if several streamlines are shown, velocity magnitude information can be inferred.

Quite a different approach to visualize 2D vector fields is described by Van Wijk [85, 86]. He uses a texture synthesis technique to visualize a 2D vector field. The texture, called spot noise, is a stochastic texture with local control. Spot noise is synthesized by addition of randomly weighted and positioned spots and local control is realized by variation of the spot. A vector field can be visualized with such a texture by varying spots according to the magnitude and direction of the local velocity. The resulting visualization gives a good visual impression of a vector field. This technique can also be used in animations, or to display a vector field on a 2D surface in 3D.

The concepts of the mappings for 2D vector fields, as described above, are easily extended to 3D. In practice, however, the effectiveness of most techniques is drastically reduced if they are applied to 3D instead of 2D vector fields. Figure 2.9 is an example of a visualization of a 3D vector field with an arrow plot technique. It is clear that, because of the large number of arrows and because of the extra
dimensions, hardly any information can be deduced from such a visualization. Indeed, the problem of having to display too much data, as it occurs in some mappings for 3D scalar fields, becomes more severe for 3D vector data.

Solutions to this problem are analogous to the solutions for 3D scalar datasets: instead of visualizing all data, only a part of the data is mapped to visual primitives. Arrow plots can be used on 2D slices of a 3D dataset, see Figure 2.10a. Addition of shadow, or projection of the arrows onto a plane, gives extra insight in the direction of the arrow. An example of this technique is described by Klassen and Harrington [42]. In the same way, the number of particle traces, streak lines and streamlines can be reduced, to avoid getting an image of cluttered lines, see Figure 2.10b. A sensible selection of starting positions, or interactively probing of the vector field, should be applied.

As in 2D, attributes of particles and streamlines may vary in 3D, either to get a better visualization, or to visualize more data. Stolk and Van Wijk [77, 86] represent particles with small facets, modelled as a point with a surface normal. These facets can be shaded with standard shading techniques, which improves the visual perception. The combined use of many of these facets can lead to visualization of streamlines, stream tubes and stream surfaces.

Hin and Post [31, 30] add a perturbation to the particle’s position, to visualize turbulent particle motion. This perturbation is based on a separate field that determines the local turbulence intensity. The resulting turbulent particle traces are visualized in an animation, or as particle traces in static images.

Ma and Smith [50] present a technique to visualize diffusion: they integrate a dispersion model in particle tracing technique. Fluid elements are traced through
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Figure 2.10: Mappings for 3D vector datasets: a) Eulerian mapping: arrow plot for a slice of a 3D dataset; b) Lagrangian mapping: a set of streamlines.

the vector field for the mean particle path, as well as the statistical dispersion of the fluid elements about the mean position, by using additional scalar information. In this way, clouds of fluid elements are traced, not just mean particle paths.

Schroeder, Volpe and Lorensen [71] describe how a regular n-sided polygon can be swept along a streamline. The polygon is oriented normal to the local velocity, and can represent local deformation due to rigid body rotation and both normal and shear strain. In addition, the effects of translation and scalar functions can be represented by appropriately varying the radius and shading of the surface of the resulting streamtube.

A volume visualization approach to particle tracing is shown by Ma and Smith [49]. From a seed position in the flow field, a volume is grown voxel by voxel, based on both the direction and magnitude of the velocity values near the seed point. This volume is visualized, and can be coloured with a scalar field, in order to visualize a scalar and vector field simultaneously. A technique that gives similar visual effects is described by Max, Becker and Crawfis [52]. They advect the vertices of a polygon that is positioned in the flow. After each advection step, the volume swept by the polygon is filled with tetrahedra. These tetrahedra are rendered using the technique described by Shirley and Tuchman [74].

The 3D analogue of a streamline is a stream surface: a surface that is tangent to a vector field. For a stationary velocity field, a stream surface amounts to the surface that is swept by a time line. A straightforward technique to generate a stream surface is tessellation of adjacent streamlines with polygons. This technique however, generally is insufficient for diverging flow, and inefficient for converging flow. Hultquist [33] describes an advancing front technique that copes with these problems. If streamlines diverge, new lines are started, and if they converge too much, lines are merged. Van Wijk [87] gives another solution for the generation of stream surfaces: a stream surface is viewed as an implicit stream
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Surface \( f(x) = C \). After the initial calculation of \( f \), a family of stream surfaces can be generated with an isosurface technique, by varying \( C \). The shape of the originating curves are defined by the value of \( f \) at the boundary of the domain. Van Wijk also describes different techniques for the evaluation of the function \( f \).

As in 2D, textures can be used for visualization of 3D vector fields. Max, Crawfis and Williams [53] describe how wind velocities above the earth’s surface can be visualized by moving cloud textures. They use a 3D fractal-like texture. After initialization, the texture coordinates are advected with the wind velocities. The texture is mapped onto cloud density contour surfaces. In this way, the are able to visualize a scalar field (cloud density) and a vector field (wind velocity) simultaneously.

In experimental flow visualization, optical flow visualization techniques are frequently applied [55]. To provide similar visualization results for numerically generated datasets, computer-generated simulations of these techniques have been developed. In this way, flow experiments can be compared with flow simulations, by comparing visualization that were created in a similar way. Mappings that simulate these optical visualization techniques are described by Pagendarm and Post [61]. Pagendarm and Post also discuss the process of comparing visualizations from different sources, such as experimental and computer graphics visualization.

A very different view on vector field visualization is given by Helman and Hesselink. In a series of papers [27, 28], they describe the development of a technique to visualize the topology of a vector field. A topological representation of a vector field consists of critical points and connecting integral curves and surfaces. The topology of a vector field gives a qualitative description of the vector field. The process of extracting the topology from a vector field consists of three steps:

- locate critical points, i.e. positions where the vector field vanishes;
- classify critical points, according to the eigenvalues of the local velocity gradient matrix;
- calculate integral curves (instantaneous streamlines) and surfaces from critical points.

Globus, Levit and Lasinski [23] give a clear description of how a topology extraction algorithm can be implemented. Topology visualization is used to visualize surfaces of separation and reattachment in fluid flows.

Silver and Samtaney et al. [76, 70] describe a feature-based approach to visualization. They extract features (“coherent amorphous regions”) from scalar and vector fields by thresholding or region-growing algorithms. Attributes are calculated for these regions, these attributes are used to identify the regions. In a time-dependent simulation, the attributes are used to track the evolution of the regions in time. The goal of this feature extraction and tracking process is to analyse the data and get to an understanding of the underlying physics.
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Another technique that also focuses on data preparation instead of visualization mapping is described by Villasenor and Vincent [81]. They developed an algorithm for the recognition and tracking of vorticity tubes, in order to visualize the evolution of these tubes in turbulent flow. A skeleton of the tube is progressively constructed using a series of joined line segments of equal length, following the geometrical axis of the tube. This skeleton is used as a starting position to find a new skeleton in the next time step. After the skeletons are located in all time steps, they are visualized by simply displaying all velocity vectors in the neighbourhood of a skeleton.

A related technique is described by Banks and Singer [3]. They also present an algorithm for the identification of vortices in complex flows. A skeleton line along the centre of a vortex is produced by a two-step predictor-corrector scheme. The technique uses a vorticity field to move in the direction of the skeleton line, and a pressure field to correct the location in the plane perpendicular to the skeleton line. They also provide for a mapping technique that displays the vortices as cylindrical objects, and maps the rotation on the vortices using grooves along the surface of these objects.

2.4.3 Tensor data mappings

In contrast to visualization of scalar and vector datasets, for which quite some mappings have been developed, the number of mappings for 3D tensor data is small. Most mappings for tensor data contain a transformation of the tensor before the data is mapped onto a visual primitive. This transformation extracts relevant information from the tensor. Real, symmetric tensors of rank two in 3D have three real eigenvalues and three real eigenvectors. As the eigenvalues and eigenvectors contain useful information, real symmetric tensors often are transformed to their eigenvalues and eigenvectors. The simplest mapping is an ellipsoid: the main axes of the ellipsoid point in the directions of the eigenvectors, and are scaled with the value of the corresponding eigenvalues. Another example is a shaft-disk: the largest eigenvalue and its corresponding eigenvector are depicted as a cylinder, the other two eigenvalues and vectors are combined into a elliptical disk. Other examples of representations of tensors, based on the decomposition of the tensor into its eigenvalues and eigenvectors, are described in papers on vector field topology visualization [23, 27].

De Leeuw [45] describes another tensor transformation, and combines it with a probing technique. His probe visualizes both velocity and velocity gradients at a position in a 3D velocity field. Instead of an eigenvalue decomposition, De Leeuw uses a decomposition into components aligned with and perpendicular to the local flow direction. A local coordinate frame is constructed, and the tensor is transformed to this coordinate system. After this transformation, the components of the tensor denote acceleration, shear, curvature, torsion and convergence/divergence of a flow field. These values are mapped to parameters of the probe, as is shown in Figure 2.11.
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Figure 2.11: Probe showing velocity and velocity gradients

Most of these tensor techniques are not effective if a whole tensor field is visualized. The mappings are, however, very useful if they are used sparingly, such as in vector field topology, or if they are combined with a probing technique, such as the probe of De Leeuw.

Delmarcelle and Hesselink describe a mapping for the visualization visualization of a tensor field [13], using the eigenvector and eigenvalue decomposition. They introduce the hyperstreamline for real symmetric tensor fields: a geometric primitive sweeps along a field line of one of the eigenvector fields, while this primitive is stretched under the combined action of the other two eigenvector fields. For structural analysis of tensor field data, they suggest an approach similar to vector field topology [14].

2.5 Rendering and Display

The final stages of the visualization pipeline are rendering and display. Visual primitives, constructed in the mapping stage, are transformed to images in the rendering stage. These images are displayed on an output device in the display stage. If the output device is a workstation screen, the display of the image is often incorporated in the rendering stage. As this is often the case, we will focus our attention on rendering of visual primitives.

Rendering techniques can be divided in two main categories: techniques for rendering simple geometric primitives, such as lines and polygons, and techniques for direct rendering of volume data. Below we will discuss parameters that con-
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

trol these rendering techniques, and the techniques for rendering visual primitives. This is followed by a discussion of techniques that render lines, polygons and volume data simultaneously (hybrid rendering techniques). This section is concluded with some statements on animation techniques.

In some sense, rendering is similar to taking a photograph: there is a scene, and a picture of it is taken from a certain viewpoint. The difference between rendering and taking a photograph is clear: in rendering (and visualization), the scene consists of a set of visual primitives that reside "in a computer", while in photography, the scene is a "real" scene. The control parameters for rendering and photography correspond. In rendering, the camera model specifies e.g. the viewing position, viewing direction, viewing distance and aperture. The position and type of light sources determines the lighting of the scene, and therefore the appearance of objects in the scene. The camera model, as well as the light source definition, may vary in complexity, depending on the rendering technique.

2.5.1 Line and polygon rendering

Rendering of visual primitives such as lines and polygons currently is a simple operation in computer graphics. Standard graphics libraries are available that support shading and Z-buffering techniques [19]. If these functions are carried out in hardware, scenes of thousands of polygons can be rendered at interactive speed. Of course, the images are not photo realistic: shadows, reflections and refractions are absent. The same visual primitives can also be rendered with more sophisticated rendering techniques. Ray tracing and radiosity techniques render images with a much higher degree of realism, at the expense of long computation times. For visualization interactivity is more important than photo realism, and therefore ray tracing and radiosity are seldom used. Only for production of high quality visualizations for presentation purposes, these techniques may be worthwhile.

2.5.2 Volume rendering

Unfortunately, it is not possible to render all visual primitives directly with simple Z-buffering techniques and shading. Rendering of volume data as 3D visual primitives (e.g. rendering a 3D scalar field as a density field, see Figure 2.12) cannot be done with a standard polygonal Z-buffer algorithm. Therefore, special techniques have been developed to render this type of data. Most of these techniques originate from medical imaging, where 3D volume datasets, such as CT and MRI datasets, are very common. Basically, there are two approaches to rendering these datasets [32]:

- forward projection: for each element of the dataset (voxel), its contribution to the image pixels is calculated and added to the image.
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

Figure 2.12: Volume rendering visualization.

- backward projection: for each pixel of the image, a colour is calculated by sampling the dataset

Forward projection techniques traverse the volume data set, and project the voxels onto the screen. The contribution of the voxel is added to, or merged with the pixel value. Voxel data sets can be traversed in front-to-back-order, in which case the front voxels are projected first, and in back-to-front-order, where projection is started with the voxels at the back of the data set. Colour merging formulas for front-to-back and back-to-front are given by Wilhelms and Van Gelder [91].

Forward projection algorithms are described by Shirley and Tuchman [74], Wilhelms and Van Gelder [91], Upson and Keeler [80] and Williams [92]. Some of these algorithms employ of graphics hardware that is capable of rendering semi-transparent polygons to achieve fast rendering. Drebin et al. [16] use a special projection technique: they resample the dataset to a new grid that is aligned with the viewing plane and the view plane normal. Rendering can then be done by simply adding all layers in the direction of the view plane normal.

Backward projection algorithms traverse the viewing plane, and calculate the colour for each pixel by casting a ray from the viewing position through the pixel into the volume data. The pixel colour is a weighted sum over the colours and opacities that are calculated at sample positions along the ray. This technique requires that the dataset is traversed along a ray for each pixel. Sabella [68] gives
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

a description of this technique. Amanatides and Woo [2] describe how Cartesian
grid can be traversed efficiently; Danskin and Hanrahan [11] also describe ways to
increase the efficiency of ray casting Cartesian volumes. Algorithms for traversing
irregular grids are described by Garrity [22] and Wilhelms and Challinger [90].

Volume rendering has been an important subject in scientific visualization
research for the past half decade. Above, we have given a very short overview of
different approaches to volume visualization, and have mentioned the main
contributions in this field. Nowadays, the focus in volume rendering is on parallel
implementations for volume rendering, and efficient techniques and special data
structures for very large datasets. To summarize these techniques is beyond
the scope of this overview. Interested readers are referred to the visualization
conference proceedings [39, 60, 40, 57, 4], and to the proceedings of the volume
visualization workshops [17, 37, 36].

2.5.3 Hybrid rendering

In scientific visualization, visual primitives of different types (such as polygonal
and volume visual primitives) must sometimes be rendered simultaneously. In
that case, a hybrid rendering technique is necessary. There are three ways to
accomplish hybrid rendering:

- convert of all types of visual primitives to a single visual primitive type
- use a general purpose rendering algorithm
- use different rendering algorithms for different types of visual primitives,
  and provide some mechanism for merging the results of the rendering

An example of hybrid rendering is shown in Figure 2.13.

The first way, conversion to a single type of visual primitives, can be im-
plemented by conversion of volume visual primitives to polygonal, or vice versa.
Conversions from volume primitives to polygonal occur in some forward projection
techniques, and can be found in [74]. Conversion of line and polygon visual
primitives to volume primitives (voxelisation) is described by Kaufman and Shi-
mony [38].

Hybrid rendering with a general purpose rendering algorithm is described by
Levoy [47]. He uses ray casting for polygonal and volume visual primitives. For
each pixel, a ray is cast through the volume, and is sampled at locations along
the ray. The colours and opacities of these samples are merged, together with the
results of casting the same ray through the set of polygonal visual primitives.

The third approach, buffering, is described by Frühauf [21] and Van Walsum
and Hin [83]. Frühauf uses two different types of renderers: a geometry renderer
and a volume renderer. Each renderer produces the same output: image space
elements (ISELs): scan conversion of polygons results in an ISEL for each pixel
covered by the polygon, ray casting of volume data results in an ISEL for every

24
sample along the ray. All ISELs are sent to a mixer module, that merges the ISELs and generates an image. Van Walsum and Hin first scan convert all lines, polygons and volume boundaries into an H-buffer, a multiple-layered Z-buffer. This buffer contains per pixel a depth-sorted list of objects that contribute to the pixel colour. After all objects are stored in this buffer, each list is traversed and the colours of the objects in the list are merged. If during merging volume boundaries are present in the list for a certain pixel, a ray is cast from that pixel through the volume data, to calculate the colour and opacity contribution of the volume data.

2.5.4 Animations

Until now, we mainly discussed rendering techniques that generate static images. Animations, however, are very useful in scientific visualization. They provide for:

- cues to enhance depth perception, for instance rotating objects (motion parallax);

- an extra display dimension, for instance slices moving through a 3D dataset;

- means for displaying dynamic data in real time.

Animations can be generated by saving a series of images on a storage medium and displaying them with a playback utility. Alternatively, if the rendering process is fast enough, the images can be rendered directly to the screen. To get an acceptable update rate (at least 10 frames per second) the visualization pipeline should be executed within 0.1 sec. If the data and mapping do not
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

change, interactive manipulation of the viewing parameters, to get better depth perception, is feasible on a workstation with graphics hardware. However, if the mapping changes, as with moving slices through the data, interactive animation is not always feasible.

The animation of time-dependent data is not trivial, especially if the images are rendered directly to the screen. If mapping and rendering times vary (and they will as the data varies), then the update rate will vary, and so will the relation between time steps in the data and real time. Van Wijk [88] addresses this problem, and shows solutions for scientific visualization animations.

2.6 Research objectives

In the previous sections, an overview of techniques for visualization is given, based on the visualization pipeline. Despite the wealth of techniques that have been developed, there are still many problems in scientific visualization for which no satisfactory solutions have been found. Below, we give a list of open research areas for scientific visualization. Then, we describe our research objectives based on these research issues, and the boundary conditions for our work. This section is concluded with an overview of the remainder of this thesis.

The following major research issues can be discerned:

- Simulation techniques can generate Mbytes, or even Gbytes of data. Efficient data management techniques are necessary to store and retrieve data from such large data sets. Also, data compression techniques that maintain relevant information in the data are necessary.

- Large datasets give problems in visualization mappings, as it is not possible to visualize all data simultaneously. Thus, information extraction by the scientist becomes difficult. Therefore, techniques are necessary that either give scientists the opportunity to select a relevant portion of the data in a flexible way, or extract the information directly from the dataset.

- There is still need for additional mappings, either to visualize the same data in another way, or to visualize new types of data. In an overview of visualization techniques for vector and tensor data, Hesselink and Delmarcelle [29] distinguish between the spatial domain (point, line, surface and volume) and the information level (elementary, local and global) of visualization techniques. After they have classified all known visualization techniques for vector and tensor data, they point out several useful combinations of spatial domain and information level that cannot be visualized yet.

- A final issue is the relation between simulation and visualization. Generally, visualization is used as a post processing tool: first a simulation is run and then the results are analysed with visualization techniques. Better
CHAPTER 2. OVERVIEW OF VISUALIZATION TECHNIQUES

modes of interaction are tracking and computational steering (see also [51]). Techniques for integrating simulation and visualization into a computational steering environment, and user interface aspects of such an integrated environment, are important research issues.

Our work focuses on the second research issue above. Our main objective is to develop general selection techniques for visualization, and mapping techniques that can be combined with these selection techniques. To achieve this, we will discuss the following subjects:

- specification of selection criteria,
- selection of relevant data,
- use of this data in selective visualization mappings,
- use of this data in iconic visualization mappings.

One of the boundary conditions is that the techniques must be applicable to 3D curvilinear grids, that are predominantly used in CFD-applications. Therefore, we also work on:

- basic visualization techniques for curvilinear grids.

The remainder of this thesis contains a description of our work in this area. In Chapter 3 we discuss basic visualization techniques for curvilinear grids. We describe a transformation technique that can be used to transform a curvilinear grid to a Cartesian grid, and show that this transformation has some problems with respect to application for point location and interpolation. Then we discuss alternative techniques for these operations, that operate directly on the curvilinear grid, and compare these techniques.

Selective visualization techniques and visualization mapping with selections are the subject of Chapter 4 and 5. In Chapter 4, we introduce a general selection technique with which data can be selected by their contents, instead of merely on their spatial position in the dataset. We show how the results of this selection technique can be used in the visualization mapping stage.

Chapter 5 builds on the results of the fourth chapter. It discusses iconic visualization, a more advanced use of selections in the visualization mapping stage. We describe how relevant parameters of selections can be calculated with volume integrals, and we describe an icon description language that can be used for binding the attributes of selections to degrees of freedom of iconic representations of selections.

In Chapter 6, we apply the techniques from Chapter 4 and 5 to some CFD datasets. In four case studies, we show that visualizations can be created efficiently and effectively with the selective and iconic visualization approach. We also show that iconic visualization can reduce the number of objects in the resulting visualization, and in that way contribute to the increase of information in a visualization.

27
Chapter 3

Visualization techniques on curvilinear grids

3.1 Introduction

Curvilinear grids are often used in computational fluid dynamics (CFD), because they have two advantages from the simulation point of view: boundary conformance can be achieved easily, even for complex geometries, and grid spacing can be varied, in order to get dense grid spacing if the solution of the simulation is expected to show large variations locally. Curvilinear grids are characterized by:

- a regular topology: in 2D, each internal grid node has four neighbours, in 3D each internal grid node has 6 neighbours,
- an irregular geometry, i.e. the positions of grid nodes generally cannot be calculated simply, but have to be enumerated.

An example of a 2D curvilinear grid is shown in Figure 2.3c.

Curvilinear grids are defined by the topology, and the positions of the grid nodes. In this chapter, we will assume that data values of datasets on curvilinear grids are specified at the grid nodes. This is not true for all grid types; staggered grids generally have vector data components specified at cell boundaries, and scalar data at cell centres. These data can, however, be interpolated to the grid nodes.

We define a grid cell as the area that is ‘in between’ four (in 2D) or eight (in 3D) grid nodes. In 2D, a cell \([i,j]\) is the polygon defined by nodes \([i,j]\), \([i,j + 1]\), \([i + 1,j + 1]\) and \([i + 1,j]\), and the cell boundaries are the straight line segments connecting these nodes, see Figure 3.1. In 3D, a cell \([i,j,k]\) is the area in between the nodes \([i + \delta,j + \delta,k + \delta]\), for \(\delta\) is 0 and 1. We define the cell boundaries of a 3D cell as bilinear surface patches through the corresponding four grid nodes.
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Figure 3.1: Definition of a 2D curvilinear grid, a grid cell and grid boundaries.

I.e., the lower $i$ boundary of 3D cell $[i, j, k]$ is the bilinear patch through the nodes $[i, j + \delta, k + \delta]$.

Unfortunately, curvilinear grids pose some problems in visualization. There are two operations frequently used in visualization algorithms, that are much more complex for curvilinear grids than for Cartesian grids: point location and interpolation.

**Point location** consists of finding the cell that contains a certain position and calculating the fractional offsets of that position within the cell found. Calculation of fractional offsets is closely related to interpolation techniques, and therefore is discussed with interpolation techniques. Finding the grid cell to which a given position belongs, is simple for Cartesian grids: if we assume that a 3D Cartesian grid is defined by an origin vector $\mathbf{o}$ and a cell size vector $\mathbf{d}$, then the position of node $\mathbf{n}$ with indices $[i, j, k]$ is given by:

$$\mathbf{n}_{ijk} = (o_x + id_x, o_y + jd_y, o_z + kd_z) \quad (3.1)$$

and consequently, the index $i$ of the cell to which a position $\mathbf{p}$ belongs can be found by:

$$i = \left\lfloor \frac{p_x - o_x}{d_x} \right\rfloor \quad (3.2)$$

and similarly for $j$ and $k$. Such a global relation for grid position does not exist for curvilinear grids. Thus, other techniques must be used for curvilinear grids. Obviously, such techniques will be computationally more expensive and more complex than the simple relation expressed by Equation 3.2.

For **interpolation**, bilinear (2D) and trilinear interpolation (3D) are generally used. A data value $v$ at a position $\mathbf{p}$ in cell $i = [i, j, k]$ with fractional offset $\alpha = (\alpha, \beta, \gamma)$ is calculated with trilinear interpolation $I_{TRI}$ as follows:

$$v_\mathbf{p} = I_{TRI,v}(i, j, k, \alpha, \beta, \gamma)$$

$$= (1 - \alpha)(1 - \beta)(1 - \gamma) v_{ijk} + \alpha(1 - \beta)(1 - \gamma) v_{i+1jk} + \alpha(1 - \beta) \gamma v_{i+1,j+1,k} + \alpha \beta(1 - \gamma) v_{i+1,j,k+1} + \alpha \beta \gamma v_{i+1,j+1,k+1}$$

30
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

\[(1 - \alpha)\beta(1 - \gamma) v_{ij+l_k} +\]
\[\alpha\beta(1 - \gamma) v_{i+l_j+k+1} +\]
\[(1 - \alpha)(1 - \beta)\gamma v_{ij+k+1} +\]
\[\alpha(1 - \beta)\gamma v_{i+j+k+1} +\]
\[(1 - \alpha)\beta\gamma v_{i+j+1+k+1} +\]
\[\alpha\beta\gamma v_{i+j+1+k+1} \]
(3.3)

The fractional offsets \(\alpha\), \(\beta\) and \(\gamma\) denote the relative position of \(p\) in the cell. For Cartesian grids, where cells have a cubical shape, these fractions can be calculated easily:

\[\alpha_p = \frac{p_{z} - (i s_{z} + o_{z})}{s_{z}}\]
(3.4)

Similar formulas hold for \(\beta\) and \(\gamma\).

The cells of a curvilinear grid generally are not cubical, therefore the offsets cannot be calculated with these formulas. As the following relation holds:

\[T_{\text{TRI},n}(i, j, k, \alpha, \beta, \gamma) = p\]
(3.5)

with \(n\) the grid node positions, the fractional offsets for curvilinear grids can be calculated by inverting the interpolation. For 2D, this results in a system of two nonlinear equations that can be solved analytically. For 3D we obtain a system of three nonlinear equations for which an analytical solution is not known. This system can be solved with a numerical technique like a Newton-Raphson iteration, but this is computationally more expensive than the computation of offsets for Cartesian grids.

In the above, we used Cartesian grids as a contrast to curvilinear grids. Note that there are more grid types that have the same properties as the Cartesian grids. All grids for which a simple, invertible, global transformation between grid position and grid cell and offset exists (such as Equation 3.2 and 3.4), can be treated in the same way as Cartesian grids. Examples of such grids are cylindrical and spherical grids.

In practice, however, the information on the global transformation is lost, or at least not available, in the visualization process. Therefore, in the visualization process, such grids are often treated as curvilinear grids, i.e. they are represented by a set of grid nodes in a regular structure, and no information on global transformation is used. This is also our approach, we will treat such grids as curvilinear grids.

In this chapter, we address point location techniques and interpolation techniques for 3D curvilinear grids. Our goal is to develop techniques and algorithms that operate in physical space, i.e. that operate in the same space as is used for the grid definition. Curvilinear grids can also be transformed to a Cartesian grid, called the computational space. This is frequently done in simulation processes such as CFD simulations. This computational space might also be used for point
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

location and interpolation. We will not use this approach here, as we want to investigate whether the use of this computational space can be avoided in the visualization process, also because this transformation has some problems, as we will show in the next section.

In Section 3.3, we describe three point location techniques, and use three different curvilinear grids to evaluate the performance of these techniques. In Section 3.4, we describe interpolation techniques, and use two curvilinear grids and a series of randomly deformed cubical cells, to evaluate the performance of the interpolation techniques. Section 3.5 gives an example of application of point location and interpolation techniques for visualization. In Section 3.6, we combine the results of Section 3.3 and 3.4.

3.2 Grid transformation

In numerical simulations it is common practice to transform a curvilinear grid to a Cartesian grid. This Cartesian grid has some advantages for numerical processing, just as a Cartesian grid has advantages for the visualization process. The output of the numerical process is generally transformed back to the curvilinear grid.

The same concept can be used in some visualization algorithms for curvilinear grids: in algorithms that integrate a streamline in a velocity field, point location and interpolation has to be done repeatedly. Such streamline algorithms might benefit from this transformation in the following way: transform the velocity field to a Cartesian grid, and do the integration on that grid, followed by a transformation of positions along the streamline back to the curvilinear grid. This last transformation is necessary, because streamlines should be visualized in physical space. Below we describe how this transformation of grid and data can be performed, where we will focus on vector data on 3D curvilinear grids.

The space in which the curvilinear grid is defined is called physical space (P-space or P), and the coordinates \( \mathbf{x} = (x, y, z) \) in \( \mathcal{P} \) are called Cartesian coordinates. The Cartesian grid is defined in computational space (C-space or \( \mathcal{C} \)), and coordinates \( \mathbf{\xi} = (\xi, \eta, \zeta) \) in \( \mathcal{C} \) are called curvilinear coordinates. The curvilinear coordinates \( \mathbf{\xi} \) can be split in an integer part \( i \) and fractional part \( \alpha \), with \( \xi = i + \alpha \). The integer part \( i = (i, j, k) \) is the cell index vector, the fractional part is the fractional offset \( \alpha = (\alpha, \beta, \gamma) \), that is used in trilinear interpolation. The transformation from \( \mathcal{C} \) to \( \mathcal{P} \) is called \( \mathcal{T} \), and the transformation from \( \mathcal{P} \) to \( \mathcal{C} \) is called \( \mathcal{T}^{-1} \) (see also Figure 3.2).

The curvilinear coordinates \( \mathbf{\xi} \) are transformed to Cartesian coordinates \( \mathbf{x} \) by trilinear interpolation over the physical space positions at the corners of a cell. This transformation is local, thus it is different for each cell. The following holds for the transformation \( \mathcal{T} \) from \( \mathcal{C} \) to \( \mathcal{P} \):

\[
\mathbf{x} = \mathcal{T}(\mathbf{\xi}) = I_{\text{TRI},n}(i, \alpha)
\]  

(3.6)
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Figure 3.2: Transformation from $\mathcal{P}$ to $\mathcal{C}$ and vice versa, for 2D curvilinear grid.

with $\mathbf{n}$ the physical positions of the grid nodes. $\mathcal{T}^{-1}$, the transformation from $\mathcal{P}$ to $\mathcal{C}$, is the inverse of this trilinear interpolation.

Vector data are transformed from $\mathcal{C}$ to $\mathcal{P}$ as follows (where all vectors are column vectors):

$$\mathbf{v}_c = \mathbf{J} \mathbf{v}_p$$

(3.7)

with $\mathbf{J}$ the Jacobian, defined by

$$\mathbf{J} = \begin{pmatrix}
\frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\
\frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\
\frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta}
\end{pmatrix}$$

$$= \frac{\partial \mathcal{T}}{\partial \xi} = \mathcal{T}'(\xi)$$

(3.8)

and similarly, for $\mathcal{P}$ to $\mathcal{C}$:

$$\mathbf{v}_p = \mathbf{J}^{-1} \mathbf{v}_c$$

(3.9)

For the application of this transformation to streamline integration, we should transform the velocity field to $\mathcal{C}$. This transformation is the crucial step in $\mathcal{C}$-space algorithms. In order to transform a vector field, we should calculate $\mathbf{J}^{-1}$, so we should calculate $\mathbf{J}$. In streamline algorithms in literature [7, 78, 73], central or forward differences are used to approximate $\mathbf{J}$ at the grid nodes. $\mathbf{J}_{ijk}$ is calculated with forward differences as follows (see also Figure 3.3a):

$$\mathbf{J}_{ijk} = \begin{pmatrix}
\mathbf{n}_{i+1,j,k,x} - \mathbf{n}_{ij,k,x} & \mathbf{n}_{i+1,j,k,z} - \mathbf{n}_{ij,k,z} & \mathbf{n}_{i+1,j,k,x} - \mathbf{n}_{ij,k,z} \\
\mathbf{n}_{i+1,j,k,y} - \mathbf{n}_{ij,k,y} & \mathbf{n}_{ij+1,k,y} - \mathbf{n}_{ij,k,y} & \mathbf{n}_{ij+1,k,y} - \mathbf{n}_{ij,k,y} \\
\mathbf{n}_{i+1,j,k,z} - \mathbf{n}_{ij,k,z} & \mathbf{n}_{ij+1,k,z} - \mathbf{n}_{ij,k,z} & \mathbf{n}_{ij+1,k,z} - \mathbf{n}_{ij,k,z}
\end{pmatrix}$$

(3.10)

with $\mathbf{n}_{ijk}$ the position of grid node $[i, j, k]$. For central differences this formula is
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

(see also Figure 3.3b):

\[
J_{ijk} = \frac{1}{2} \begin{pmatrix}
(n_{i+1,jk,x} - n_{i-1,jk,x}) & (n_{ij+1,k,x} - n_{ij-1,k,x}) & (n_{ijk+1,x} - n_{ijk-1,x}) \\
(n_{i+1,jk,y} - n_{i-1,jk,y}) & (n_{ij+1,k,y} - n_{ij-1,k,y}) & (n_{ijk+1,y} - n_{ijk-1,y}) \\
(n_{i+1,jk,z} - n_{i-1,jk,z}) & (n_{ij+1,k,z} - n_{ij-1,k,z}) & (n_{ijk+1,z} - n_{ijk-1,z})
\end{pmatrix}
\] (3.11)

Another formula for \( J \) is obtained if we substitute the interpolation formula from Equation 3.3 in Equation 3.8. For node \([i, j, k] \) of cell \([i, i, k] \), we should fill in (0,0,0) for \( \alpha \), for node \([i, j, k+1] \) we should fill in (0,0,1) for \( \alpha \), etc. It then follows that we get a different formula for \( J \) for each node of the cell. Figure 3.3c shows how a Jacobian is calculated for the upper left node of a 2D cell: a combination of forward \( (J_1) \) and backward \( (J_2) \) differences is used. In Figure 3.3d the Jacobians for the lower right node of a 2D cell are shown: a combination of backward \( (J_1) \) and forward \( (J_2) \) differences. Likewise, forward differences should be used for the lower left node, and backward differences for the upper right node of the cell. In this way, all differences are calculated with the positions of the corner nodes of the cell, data of other nodes are not used (as is the case with standard forward, or central differences). The formula for the 3D Jacobian for node \([i, j, k] \) of cell \([l, m, n] \) (with \( l \leq i \leq l+1 \), and similar for \( j \) and \( k \)) is given by:

\[
J_{ijk} = \begin{pmatrix}
(n_{i+1,jk,x} - n_{i-1,jk,x}) & (n_{im+1,k,x} - n_{imk,x}) & (n_{ijn+1,z} - n_{ijn,z}) \\
(n_{i+1,jk,y} - n_{i-1,jk,y}) & (n_{im+1,k,y} - n_{imk,y}) & (n_{ijn+1,y} - n_{ijn,y}) \\
(n_{i+1,jk,z} - n_{i-1,jk,z}) & (n_{im+1,k,z} - n_{imk,z}) & (n_{ijn+1,z} - n_{ijn,z})
\end{pmatrix}
\] (3.12)

The way Jacobians are calculated is important for the application of this transformation in streamline algorithms. Sadarjoen et al. [69] show that the last type of Jacobian calculation indeed gives better results for streamline integration. The streamlines that are calculated for a velocity field that is transformed with these Jacobians approximate analytic solutions (and physical space solutions) better than streamlines calculated for velocity fields that were transformed with forward or central differences Jacobians.

However, mixed-type differences also have a problem. Each node will have eight different transformed vectors, as the type of differencing (forward or backward) depends on the cell to which a node belongs. Thus, a transformed vector field in \( \mathcal{C} \) generally is discontinuous at cell boundaries. This was to be expected: as \( \mathcal{T} \) is only \( C_0 \)-continuous, \( J \) (and \( Jv \)), being a derivative of \( \mathcal{T} \), may be discontinuous. Sadarjoen et al. [69] state that this discontinuity may cause substantial errors if the grid deformation is relatively large.

It should be noted that these transformation problems might be solved in other ways. Numerical processes that operate in \( \mathcal{C} \) generally use staggered grids. The results of the numerical process are transformed to \( \mathcal{P} \) and interpolated to grid nodes before they are output. Then, in the streamline integration, the data are transformed to \( \mathcal{C} \) again. Of course, it would be much better if the results
Figure 3.3: Jacobian calculation for 2D grid, $J_1$ is $\Delta x / \Delta \xi$, $J_2$ is $\Delta x / \Delta \eta$, nodes used for Jacobian calculation are shown, and for c) and d) the cell to which the Jacobian applies is depicted in gray: a) forward differences; b) central differences; c) mixed differences, corresponding cell is lower-right cell; d) mixed differences, corresponding cell is upper-left cell.
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

in \( \mathcal{C} \) (even on staggered grids) could be fed directly in a streamline algorithm for \( \mathcal{C} \), instead of interpolating and transforming the data twice. In that case, the transformation problem would be reduced to transforming positions in \( \mathcal{C} \) to positions in \( \mathcal{P} \), a simple trilinear interpolation. Thus techniques for \( \mathcal{C} \) could be very efficient if they could handle staggered grids, and if they could get their \( \mathcal{C} \)-data directly from the numerical simulation.

However, it is common practice that grid data are output at nodes of a curvilinear grid in \( \mathcal{P} \), because post processing tools such as visualization programs expect them to be in \( \mathcal{P} \). Therefore, in order to avoid these transformation problems, we decided to investigate algorithms for curvilinear grids that operate directly in \( \mathcal{P} \).

3.3 Point location in \( \mathcal{P} \)

In Section 3.1 it has been discussed that point location is an operation that is considerably more difficult for curvilinear grids than for Cartesian grids. The point location problem can be stated very shortly: determine the cell which contains an arbitrary target position \( \mathbf{p} \) in a curvilinear grid, i.e. inside which cell is position \( \mathbf{p} \)? (Again, we postpone fractional offset calculation to the section on interpolation.) This point location problem occurs in different ways in visualization algorithms. For the exploration of a dataset with a probe, the data values at the probing position must be calculated, and therefore it must be known in which cell the probe is. Another example is the calculation of a streamline, for which the cell of the starting position must be found, and for each integration step, the cell in which the next position is located, must be found.

Note that point location is necessary to calculate a data value at an arbitrary position in the grid. As data calculation is done with interpolation techniques, some techniques in this section are related to techniques in the section on interpolation.

Two types of point location can be distinguished in visualization:

- global point location, for example finding the cell for a starting position of a streamline,

- local point location, for example finding the cells for the new position calculated by of an integration step in streamline integration; in this case there is extra information in the form of the cell containing the previous position.

The global point location problem can be approached in three ways. First of all, there is the brute force approach: just try each cell, until the right cell is found. This is, of course, computationally very expensive. Second, there is the possibility to introduce extra global datastructures, as described by Neeman [56]. Thirdly, global point location problems can be reduced to local point location problems by choosing an arbitrary starting position and cell. Local point location problems can be solved by performing a local search for the right cell.
We focus on local point location techniques, which use the position in a known cell as a starting position for the search for the target position. From the starting position, the grid is traversed by stepping from cell to cell, in the direction of the target position. In the following, we describe three ways of stepping through the curvilinear grid to the target position.

### 3.3.1 Stencil Walk

The Stencil Walk algorithm is described by Buning [7]. It is an iterative algorithm that uses computational space for finding the cell and offsets for a physical space position. This amounts to finding the computational space coordinates of a physical space position: the integer part of the computational space coordinates is the cell index vector, and the fractional part is the offset. Let $p$ be a point in physical space that must be found in computational space. First an initial point $\alpha$ in some cell $i$ in computational space is chosen. This point is transformed to physical space, using the transformation from Equation 3.6:

$$x = T(i, \alpha)$$

(3.13)

The difference between the transformed and the target point $p$ is calculated as:

$$\Delta x = x - p$$

(3.14)

and this difference vector in physical space is transformed back to computational space, using Equation 3.9:

$$\Delta \alpha = T'(i, \alpha)^{-1} \Delta x$$

(3.15)

The difference vector $\Delta \alpha$ is added to the previous position, resulting in a new guess:

$$\alpha_{n+1} = \alpha_n + \Delta \alpha_n$$

(3.16)

If one of the elements of $\alpha_{n+1}$ is outside the range $[0, 1]$, the centre of the corresponding neighbouring cell is the new guess. The iterative process continues until the right cell is found. If the iterative process is continued until each element of $\Delta \alpha_n$ is smaller than some epsilon value, $\alpha_n$ contains the fractional offsets of $p$ in the corresponding cell. Buning states that this algorithm might have problems at grid singularities, or at grid boundaries. Also, he suggests to use a set a points spread throughout the grid as starting positions for an global point location with the Stencil Walk algorithm.

If we assume that all elements of $\alpha_n$ and $\alpha_{n+1}$ are within the range $[0, 1]$, we can combine Equations 3.13, 3.14, 3.15 and 3.16 to the following formula for $\alpha_{n+1}$:

$$\alpha_{n+1} = \alpha_n + T'(i, \alpha_n)^{-1} (T(i, \alpha_n) - p)$$

(3.17)

If we introduce

$$F = p - T(i, \alpha_n)$$

(3.18)
and the derivative of $F$, being

$$F' = -T'(i, \alpha_n)$$

we can rewrite Equation 3.17 as:

$$\alpha_{n+1} = \alpha_n - \frac{F(\alpha_n)}{F'(\alpha_n)}$$

which is the formula for a Newton-Raphson iteration. The Stencil Walk algorithm thus is a Newton-Raphson iteration, extended with a test whether the elements of $\alpha_{n+1}$ are within the range $[0,1]$.

A property of the Stencil Walk algorithm is that it does not traverse the grid in a straight line. Figure 3.4a shows a problem that can occur if the grid is not traversed in a straight line: the target point is not reached because the algorithm leaves the grid (the algorithm steps outside the grid). Therefore, we adapted the Stencil Walk algorithm, to provide for stepping along grid boundaries. As long as at least one element of $\alpha_{n+1}$ does not cross a grid boundary, the search is
stencil_walk_cell(alpha, cell, target)
{
    alpha_n1 = initialize_alpha();
    do {
        alpha_n = alpha_n1;
        x = interpolate_pos(cell, alpha_n);
        delta_x = target - x;
        J = interpolate_jacobian(cell, alpha_n);
        J_inv = invert(J);
        delta_alpha = J_inv * delta_x;
        alpha_n1 = alpha_n + delta_alpha;
        outside = check_ranges(alpha_n1);
    } until ( |delta_alpha| < epsilon || outside )
    if (outside)
        return(alpha_n1);
    else
        return(alpha_n);
}

Figure 3.5: Pseudocode for Stencil-Walk routine.

continued in the corresponding neighbouring cell. A resulting traversal is shown in Figure 3.4b.

The implementation is straightforward; there is a routine for performing an iteration within a cell, and a routine that takes care of stepping to the next cell. For each new cell entered, local values such as the grid coordinates of the cell and the Jacobians at the corner nodes, are stored, so that these values are calculated only once for each cell visited. Pseudo-code for the first routine, that implements Equations 3.13–3.16, is shown in Figure 3.5. A visualization of the cells traversed by the Stencil Walk algorithm is shown in Figure 3.6.

3.3.2 Bilinear cell boundaries

The Stencil-Walk algorithm is an iterative search technique, based on the Newton-Raphson iteration. An alternative approach for local point location is to use
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Figure 3.6: Cells traversed with Stencil Walk algorithm (traversal direction is from right to left).

generic techniques. If we draw a line from the position of which the cell index vector is known to the unknown position, we can walk along the line, thereby detecting when a cell is left, and which new cell is entered. At the end of the line, the cell index vector for the target position will be known.

In order to use such a geometric approach, we should be able to detect when a cell is left, and a new cell is entered. This amounts to calculating the intersection of the cell boundaries with the line. In Section 3.1, cell boundaries of 3D curvilinear grid cells were defined as bilinear patches. To detect where a line leaves a cell, we have to intersect the bilinear cell boundaries with a line. The equation of a bilinear patch $p$ through nodes $n_{00}$, $n_{01}$, $n_{10}$ and $n_{11}$ is:

$$p = (1 - \alpha)(1 - \beta)n_{00} + (1 - \alpha)\beta n_{01} + \alpha(1 - \beta)n_{10} + \alpha\beta n_{11}$$  \hspace{1cm} (3.21)

and the equation for a line $l$ from point $s$ to $t$ is:

$$l = s + \lambda(t - s), \quad \text{with } 0 \leq \lambda \leq 1$$  \hspace{1cm} (3.22)

or, with $v = t - s$,

$$l = s + \lambda v, \quad \text{with } 0 \leq \lambda \leq 1$$  \hspace{1cm} (3.23)

The intersection follows from the combination of Equation 3.21 and 3.23:

$$(1 - \alpha)(1 - \beta)n_{00} + (1 - \alpha)\beta n_{01} + \alpha(1 - \beta)n_{10} + \alpha\beta n_{11} = s + \lambda v$$  \hspace{1cm} (3.24)

which can be rewritten as:

$$n_{00} - s + (n_{10} - n_{00})\alpha + (n_{01} - n_{00})\beta + (n_{00} - n_{01} - n_{10} + n_{11})\alpha\beta = \lambda v$$  \hspace{1cm} (3.25)
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

This system of equations can be solved analytically, with the following possible solutions:

- no solutions, thus no intersection
- one solution, the line intersects the patch once, or it grazes the patch
- two solutions, the line intersects the patch twice
- more than two solutions, the line coincides with a line of the bilinear patch, or two or more nodes coincide and the line intersects the patch at those nodes.

For determining when a cell is left, the only relevant cases are one or two intersections, in the other cases the cell is not left.

This traversal algorithm has been implemented with two routines, one for calculating the intersection of the line with the cell boundaries, and one for stepping to the next cell. The first routine intersects the line with all cell boundaries, until a valid intersection is found, or until all cell boundaries are tried. An intersection is considered a valid intersection if it satisfies the following conditions:

- \( \alpha \) and \( \beta \) are within \([0, 1]\) (the intersection is inside the patch),
- \( \lambda \) at the intersection is between \( \lambda \) at the previous intersection and \( \lambda \) at the end of the line (the intersection is at the right part of the line),
- the dot product of the line direction \( \mathbf{v} \) and the outward bounded normal of the patch at \((\alpha, \beta)\) is positive (thus the cell is left at the intersection). The outward bounded normal can be calculated by taking the cross product of the partial derivatives of Equation 3.21 with respect to \( \alpha \) and \( \beta \).

To allow for small numerical errors in the intersection calculation (in \( \alpha \) and \( \beta \)) we keep track of an intersection that is "almost" valid, e.g. an intersection with the bilinear patch just outside the cell boundaries (\( \alpha \) or \( \beta \) > 1, or < 0). If no valid intersection is found, but an almost valid intersection is found, this intersection is used. An algorithm that implements this is shown in Figure 3.7.

The intersection calculation is computationally expensive. Therefore, we added a bounding box test to the intersection calculation. A bounding box is created around the four corner positions of a patch, and the line segment from the last intersection position to the target position is compared with the bounding box. If the bounding box of the line segment is outside of the bounding box the patch, the bilinear patch intersection is skipped.

An example of the cells visited by this algorithm is shown in Figure 3.8.
bilinear_patch_cell(line,cell)
{
    for (all cell boundaries patch) {
        calculate_intersections(patch,line,isects);
        for (all intersections isect) {
            if (valid_intersection(isect));
                return(isect);
            else if (almost_valid_intersection(isect));
                almost_valid_intersection = isect;
        }
    }
    if (almost_valid_intersection found)
        return(almost_valid_intersection)
    else
        return(no_intersection)
}

Figure 3.7: Pseudocode for bilinear cell boundary stepping routines.

Figure 3.8: Cells traversed with bilinear cell boundaries.
3.3.3 Tetrahedrization

The bilinear patch approach, described in the previous section, has a disadvantage: the intersection calculations are computationally expensive. Also, in most curvilinear grids, the cell boundaries will be almost planar, in which case a bilinear patch may not be necessary to approximate the cell boundaries.

An alternative approach is to subdivide each cell boundary into two planar triangles, see Figure 3.9a. Determining the boundary through which the line leaves the cell amounts to intersecting the line with the twelve triangles of the cell boundaries. The intersection of the line and the triangle is valid if the line can leave the cell at the triangle, i.e. if the dot product of the line direction and the outward pointing triangle normal is positive. For this approach, the intersection point on the parametrized triangle has to be calculated to determine whether the intersection point is inside the triangle. This approach is faster than the bilinear patches, but still is computationally expensive.

The same idea of triangulation of the cell boundaries can be used if we decompose a hexahedral cell into five tetrahedra, as shown in Figure 3.9b. This decomposition can be done in two ways. For a grid, an alternating decomposition should be applied, to ensure that the triangles at the cell boundaries match the triangles of the neighbouring cells, see Figure 3.10.

Such tetrahedral decompositions have been applied successfully for volume ray casting on curvilinear grids [22, 74]. For general point location problems, they can also be used. Stepping through hexahedral cells then becomes stepping through tetrahedra. An advantage of tetrahedral traversal, compared to traversal using triangular decomposition of cell boundaries, is that it is not necessary to calculate the positions of the intersections of a line with the faces of the tetrahedra. It suffices to calculate $\lambda$ at the intersection of the line with each face of a tetrahedra,
and this is only necessary for those faces of a tetrahedron where the line might leave the tetrahedron. With line $l$ defined according to Equation 3.23, and the face $t$ of a tetrahedron, defined by the plane through that face:

$$\mathbf{n}_t \cdot \mathbf{x} + d_t = 0$$  \hspace{1cm} (3.26)

with $\mathbf{n}_t$ the outward pointing plane normal, $\mathbf{x}$ a position on the plane and $d_t$ the plane constant, the intersection lambda $\lambda_{isect}$ can be calculated with:

$$\lambda_{isect} = \frac{-d_t - s \cdot \mathbf{n}_t}{\mathbf{v} \cdot \mathbf{n}_t}$$  \hspace{1cm} (3.27)

The tetrahedral traversal algorithm simply calculates the intersection $\lambda$ for each face of which $\mathbf{v} \cdot \mathbf{n}_t$ is positive, and chooses the intersection with the smallest $\lambda$, see the pseudocode in Figure 3.11. From the current tetrahedron and the face through which the tetrahedron is left, it can easily be determined whether stepping to a next tetrahedron in the same hexahedral cell is required, or whether the cell is left. An example of the tetrahedra visited with this algorithm is shown in Figure 3.12.

### 3.3.4 Comparison

The efficiency of the traversal algorithms described in the previous sections has been compared. Each algorithm has been implemented in the same way: a routine for stepping to the next cell, and a routine for stepping through the grid to the end of a line. For the tests, we used three different grids: a cylindrical grid, the grid of the backward facing step, and the grid of the blunt fin (see Figure 3.13). The tests were performed on three different types of SGI workstations: a Personal Iris (R3000 processor at 20 MHz), an Indy (R4600 processor at 134 MHz) and an Onyx (R8000 processor at 75 MHz).

Our goal is to determine the relative efficiency of these algorithms. As it is to be expected that the algorithms perform differently for global and local point
tetrahedral-cell(line,tetra,cell)
{
    do {
        calculate_intersected_face(tetra,line,face)

        if (no face intersected)
            return(no_intersection)
        else
            calculate_new_tetra(tetra,face)
    } until (cell is left)

    return(face)
}

Figure 3.11: Pseudocode for tetrahedral traversal routines.

Figure 3.12: Tetrahedra traversed with a tetrahedral decomposition.
Figure 3.13: Grids for comparing efficiency of point location algorithms: a) cylindrical; b) backward facing step; c) blunt fin
location (i.e. for large and small distances between starting and target position), we varied the number of steps along the line from the start to the target position, i.e. we called the stepping routine once for the whole line, or twice (once for the first half part of the line, and once for the second half), or ten times (with the line split in ten parts of equal length) etc. As stepping is performed along a straight line, repeated local point location is equivalent to streamline tracing in a constant velocity field (thus without interpolation). Stepping along a line through a grid, thereby determining which cells are intersected by the line, is comparable to ray casting of volume data on a curvilinear grid.

In Figures 3.14–3.16 results are shown: for each grid the statistics for a representative traversal are displayed. From these figures, it is clear that the bilinear patch approach is rather inefficient. The Stencil Walk approach is efficient for global point location, and the tetrahedral approach is efficient if the number of steps along the line is high. The inefficiency of the bilinear patch approach is caused by the computationally expensive line-patch intersection calculation. The Stencil Walk algorithm performs well if the line is rather long. This is because the algorithm only needs one iteration per cell to decide to step to the next cell. However, if the line is small compared to the cell size, the Stencil Walk algorithm needs several iterations to locate the point within the current cell. Therefore, the number of iterations increases rapidly if the line is cut into smaller parts. For the tetrahedral algorithm, the number of tetrahedra visited is constant. The increase in traversal time that results from traversing smaller line parts with the tetrahedral traversal algorithm is less than the increase for the Stencil Walk algorithm,

![Graph](image)

**Figure 3.14:** Results for traversal algorithm efficiency test on cylindrical grid, line 3, showing the traversal time (in sec.) per number of steps along the line (for SGI Onyx).
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Figure 3.15: Results for traversal algorithm efficiency test on backward facing step grid, line 8, showing the traversal time (in sec.) per number of steps along the line (for SGI Indy).

Figure 3.16: Results for traversal algorithm efficiency test on blunt fin grid, line 4, showing the traversal time (in sec.) per number of steps along the line (for SGI Personal Iris).
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

because it is simpler to detect that a line segment does not leave a tetrahedron, than to perform a Newton-Raphson iteration.

In the algorithms, we counted the number of cells that were traversed by the stepping algorithms. With these data, we can calculate the average number of steps taken per cell. For each test case, we calculated the position where the Stencil Walk and the tetrahedral traversal performed equally well, in steps per cell. These numbers are shown in Table 3.1–3.3. From these tables, it is clear that Stencil Walk and tetrahedral traversal perform equally well if the number of steps per cell is around 0.8. For streamline integration with second order Runge Kutta (i.e. two steps per integration step), this means that at 0.4 integration steps per cell, the algorithms perform equally well. If more steps per cell are taken, tetrahedral traversal performs better, otherwise Stencil Walk performs better. As streamline integration generally needs several integration steps per cell, tetrahedral traversal should be preferred for streamline integration.

Note that, in the above, we do not account for interpolation techniques. For trilinear interpolation, we have to calculate fractional offsets of a position in a cell. Generally, an iterative technique like Newton-Raphson is used for this purpose. If we use such an iterative technique in combination with tetrahedral traversal, it will perform worse than the Stencil Walk, as these fractions are calculated as part of the Stencil Walk algorithm. The next section will discuss these interpolation issues in relation to the traversal algorithms in more detail.

Apart from the efficiency, we also looked at the robustness of the algorithms. In the grids used, both the tetrahedral and bilinear patch algorithms did not have any problems, as long as the line did not intersect the grid boundaries. The Stencil Walk also performed well, and was even able to traverse the grid if the line between start and target positions did intersect the grid boundary. The difference between the Stencil Walk and the geometric algorithms is shown in Figure 3.17. Figure 3.17a shows the cells traversed by the Stencil Walk algorithm, and it shows that the algorithm is capable of stepping along grid boundaries if

<table>
<thead>
<tr>
<th>line nr</th>
<th>from cell</th>
<th>to cell</th>
<th>cells/line</th>
<th>St. Walk = tetrah. steps</th>
<th>steps/cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,0,0]</td>
<td>[29,0,0]</td>
<td>30</td>
<td>21</td>
<td>0.68</td>
</tr>
<tr>
<td>2</td>
<td>[0,0,0]</td>
<td>[0,0,29]</td>
<td>30</td>
<td>30</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>[0,0,0]</td>
<td>[29,8,0]</td>
<td>38</td>
<td>27</td>
<td>0.72</td>
</tr>
<tr>
<td>4</td>
<td>[29,1,0]</td>
<td>[29,17,0]</td>
<td>73</td>
<td>58</td>
<td>0.79</td>
</tr>
<tr>
<td>5</td>
<td>[29,1,0]</td>
<td>[29,17,29]</td>
<td>102</td>
<td>80</td>
<td>0.78</td>
</tr>
<tr>
<td>6</td>
<td>[29,17,29]</td>
<td>[29,1,0]</td>
<td>102</td>
<td>83</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Table 3.1: Start and target positions for traversal tests in the cylindrical grid, the number of cells along the line and the number of steps (total and per cell) at which Stencil Walk and tetrahedrization perform equally well (for SGI Onyx).
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

<table>
<thead>
<tr>
<th>line nr</th>
<th>from cell</th>
<th>to cell</th>
<th>cells/line</th>
<th>St. Walk = tetrah. steps</th>
<th>steps/cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,0,0]</td>
<td>[0,35,0]</td>
<td>36</td>
<td>34</td>
<td>0.93</td>
</tr>
<tr>
<td>2</td>
<td>[0,0,0]</td>
<td>[23,0,0]</td>
<td>24</td>
<td>22</td>
<td>0.92</td>
</tr>
<tr>
<td>3</td>
<td>[0,0,0]</td>
<td>[0,0,7]</td>
<td>8</td>
<td>8</td>
<td>0.98</td>
</tr>
<tr>
<td>4</td>
<td>[11,24,4]</td>
<td>[0,0,0]</td>
<td>40</td>
<td>33</td>
<td>0.83</td>
</tr>
<tr>
<td>5</td>
<td>[11,24,4]</td>
<td>[0,35,0]</td>
<td>27</td>
<td>23</td>
<td>0.85</td>
</tr>
<tr>
<td>6</td>
<td>[11,24,4]</td>
<td>[0,35,7]</td>
<td>26</td>
<td>22</td>
<td>0.83</td>
</tr>
<tr>
<td>7</td>
<td>[11,24,4]</td>
<td>[0,0,7]</td>
<td>39</td>
<td>33</td>
<td>0.84</td>
</tr>
<tr>
<td>8</td>
<td>[11,24,4]</td>
<td>[23,0,0]</td>
<td>41</td>
<td>33</td>
<td>0.80</td>
</tr>
<tr>
<td>9</td>
<td>[11,24,4]</td>
<td>[23,0,7]</td>
<td>40</td>
<td>33</td>
<td>0.82</td>
</tr>
<tr>
<td>10</td>
<td>[11,24,4]</td>
<td>[23,18,0]</td>
<td>23</td>
<td>14</td>
<td>0.60</td>
</tr>
<tr>
<td>11</td>
<td>[11,24,4]</td>
<td>[23,35,0]</td>
<td>28</td>
<td>21</td>
<td>0.77</td>
</tr>
<tr>
<td>12</td>
<td>[11,24,4]</td>
<td>[23,35,7]</td>
<td>27</td>
<td>19</td>
<td>0.72</td>
</tr>
<tr>
<td>13</td>
<td>[11,24,4]</td>
<td>[23,18,7]</td>
<td>22</td>
<td>14</td>
<td>0.64</td>
</tr>
</tbody>
</table>

Table 3.2: Start and target positions for traversal tests in the backward facing step grid, the number of cells along the line and the number of steps (total and per cell) at which Stencil Walk and tetrahedrization perform equally well (for SGI Onyx).

<table>
<thead>
<tr>
<th>line nr</th>
<th>from cell</th>
<th>to cell</th>
<th>cells/line</th>
<th>St. Walk = tetrah. steps</th>
<th>steps/cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,0,0]</td>
<td>[0,30,30]</td>
<td>61</td>
<td>40</td>
<td>0.66</td>
</tr>
<tr>
<td>2</td>
<td>[0,0,0]</td>
<td>[0,30,0]</td>
<td>31</td>
<td>19</td>
<td>0.61</td>
</tr>
<tr>
<td>3</td>
<td>[0,0,0]</td>
<td>[0,0,30]</td>
<td>31</td>
<td>24</td>
<td>0.76</td>
</tr>
<tr>
<td>4</td>
<td>[10,14,0]</td>
<td>[38,30,30]</td>
<td>77</td>
<td>77</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 3.3: Start and target positions for traversal tests in the blunt fin grid, the number of cells along the line and the number of steps (total and per cell) at which Stencil Walk and tetrahedrization perform equally well (for SGI Onyx).
the target position cannot be reached along a straight line. Figure 3.17b shows that geometric algorithms cannot find the target position.

This feature of the Stencil Walk algorithm is advantageous for global point location, as with this algorithm target positions might be found that cannot be reached from the starting position by a straight line. Note that the same feature might be a disadvantage for streamline integration, as we do not want to integrate streamlines if the line leaves the grid (even if it reenters the grid). The geometric approaches have the advantage that they always detect if a line leaves the grid. However, as in streamline integration the integration steps will be relatively small, cases as shown in Figure 3.17a will not occur.

A final word on badly shaped grids. We were able to construct grids where each of the algorithms failed, even in streamline integration. Stencil Walk algorithms can start iterating between cells, tetrahedral decomposition of badly deformed cells may result in intersecting cells, and even bilinear patches appeared to fail in such grids. In this case, however, the grid was not a grid that resulted from a 3D numerical simulation. In practice, these problems will not occur as the grids that result from numerical simulations behave well, as this is a necessary condition for the application of numerical techniques.


CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

3.4 Interpolation in \( \mathcal{P} \)

In the previous section, we showed that point location can be performed successfully in \( \mathcal{P} \). In this section, we describe how to interpolate in \( \mathcal{P} \). Again, we will focus on the 3D case.

In the following, we describe three ways to calculate interpolated values for a dataset: trilinear interpolation in a hexahedron, linear interpolation in a tetrahedron and trilinear interpolation via tetrahedrization. The last technique combines the first two techniques to approximate a trilinear interpolation at low cost. After this description, we discuss the possibilities of applying these techniques in streamline tracing and ray casting algorithms, based on a global comparison of the efficiency and accuracy of the interpolation techniques.

3.4.1 Interpolation techniques

For trilinear interpolation (TRILIN) of a value \( v \) at position \( p \) in a hexahedron, with \( \alpha_h, \beta_h \) and \( \gamma_h \) the fractional offsets of \( p \), the following formula holds (see also Equation 3.3):

\[
v_p = I_{\text{TRI},v}(\alpha_h, \beta_h, \gamma_h) = (1 - \alpha_h)(1 - \beta_h)(1 - \gamma_h)v_{000} + \\
\alpha_h(1 - \beta_h)(1 - \gamma_h)v_{100} + \\
(1 - \alpha_h)\beta_h(1 - \gamma_h)v_{010} + \\
\alpha_h\beta_h(1 - \gamma_h)v_{110} + \\
(1 - \alpha_h)(1 - \beta_h)\gamma_h v_{001} + \\
\alpha_h(1 - \beta_h)\gamma_h v_{101} + \\
(1 - \alpha_h)\beta_h\gamma_h v_{011} + \\
\alpha_h\beta_h\gamma_h v_{111}
\]  

(3.28)

with \( v_{ijk} \) the value at corner node \( n_{ijk} \) of the hexahedron.

For this interpolation, we must know the fractional offset \( \alpha_h \) for position \( p \). If these fractions are unknown, they can be calculated with a Newton-Raphson iteration, as described in Section 3.2. The following techniques avoid this relatively expensive iteration, either by not using the offsets, or by a cheaper approximation of these offsets.

If we use a tetrahedrization of hexahedrons, this tetrahedrization can also be used for interpolation. Linear interpolation of a value \( v \) at position \( p \) in a tetrahedron with vertices \( A, B, C \) and \( D \) is given by

\[
v_p = v_D + \alpha_t(v_A - v_D) + \beta_t(v_B - v_D) + \gamma_t(v_C - v_D)
\]  

(3.29)

where \( \alpha_t = (\alpha_t, \beta_t, \gamma_t) \) are the offsets in the tetrahedron, determined by the linear parametrization of the tetrahedron:

\[
P_{ABC,\alpha_t\beta_t\gamma_t} = D + \alpha_t(A - D) + \beta_t(B - D) + \gamma_t(C - D)
\]  

(3.30)
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

with

\[ \alpha_t, \beta_t, \gamma_t \geq 0 \quad \text{and} \quad 0 \leq \alpha_t + \beta_t + \gamma_t \leq 1 \]

The value for \( \alpha_t \) can be determined by:

\[
\alpha_t = \begin{pmatrix} A - D & B - D & C - D \end{pmatrix}^{-1} \begin{pmatrix} P - D \end{pmatrix} \quad (3.31)
\]

This technique (TetLin) requires one matrix inversion, two vector-matrix multiplications, and some vector subtractions to calculate an interpolated value.

Trilinear interpolation in a hexahedron and linear interpolation in a tetrahedron can be combined (TetTri). If a hexahedron is tetrahedrized, the offsets \( \alpha_t \) in a tetrahedron can be calculated in the way described above. As the values of the offsets \( \alpha_h \) are known at the vertices of the tetrahedron, they can be linearly interpolated in the tetrahedron, yielding an approximation \( \alpha'_h \) of the actual offsets \( \alpha_h \). This approximated offset \( \alpha'_h \) can be used in a trilinear interpolation formula. Note that, if we apply a tetrahedrization as described in Section 3.3.3, and if we choose the right parametrization for the tetrahedra at the corners of the hexahedron, the transformation from \( \alpha_t \) to \( \alpha'_h \) for these tetrahedra consists of up to three scalar additions or subtractions, instead of a linear interpolation.

### 3.4.2 Comparison

Interpolation techniques determine the data that is to be visualized. Therefore, if we use another interpolation technique, we might get different results in the visualization. In some cases, this does not change the final visualization result, in other cases, it might considerably change the final result. To determine when we are allowed to use certain interpolation techniques, we did the following experiments:

- interpolation errors of TetLin and TetTri (with respect to Trilin, which is used as a reference) were calculated and compared,

- streamline integration errors for second order Runge Kutta integration were compared with the TetLin and TetTri interpolation errors for two CFD datasets,

- timing results were gathered to account for the computational expenses of the interpolation techniques.

In the following, we will describe these experiments and the results.

Before we describe these tests, however, it is important to note that the origin of the dataset is a criterion for choosing the interpolation technique. The assumptions made in a numerical simulation determine the use of the data in the visualization. If, in a simulation, the data is assumed to vary linearly in a cell, (tri)linear interpolation should be used for visualization. Using higher order interpolation techniques will suggest a higher order continuity, which is not
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

justified. In our test cases, the data is assumed to be \( C_0 \) continuous, therefore higher order interpolation techniques are not necessary. Furthermore, if the data is assumed to vary (tri)linearly in a cell, trilinear interpolation can be viewed as the appropriate interpolation.

Two observations can be made regarding the relation of the techniques presented above. First, if the values at the vertices of a hexahedron are a linear function of their position, then TETLIN and TRIHIN will yield identical results, while TETTRI may give a result that differs from TETLIN. Second, if the hexahedron is a parallelepiped, TETTRI will yield a result equal to standard TRIHIN, while TETLIN may give a result that differs from TRIHIN, depending on the distribution of the values at the grid nodes. More generally, we can say that the relation between TRIHIN and TETLIN depends on the distribution of the data values at the grid nodes, while the relation between TETTRI and TETTRI depends on the shape of the hexahedron and the gradient of the data values. If the error in the approximated fractional offsets is \( \Delta \alpha \), then the final interpolation error is a linear function of \( \Delta \alpha \), determined by the value of the data gradient.

Quantitative results of a comparison between the interpolation techniques were obtained by interpolating data with these techniques, and comparing the relative interpolation errors of TETTRI and TETLIN. We did this for two cases:

- cells of two CFD data sets
- randomly generated cells, with the values at the nodes determined by a function of the node position

For the first tests, we used data from two different CFD datasets: a flow in a backward facing step geometry, and a flow around a blunt fin. In each cell, we choose five random positions, and interpolated the data values in the cell at those positions. For the backward facing step data set (bfs), we have four data fields: pressure and three velocity components. For the blunt fin data set (fin) we have five data fields: density, stagnation and momentum. For each position, we calculated the relative interpolation error for TETTRI and TETLIN with respect to TRIHIN. We also compared the relative errors at each position, and calculated for how many positions the TETTRI interpolation error was greater than the interpolation error for TETLIN. The average and standard deviation of the relative interpolation errors, and the fraction of samples at which the relative interpolation error of TETTRI is greater than the interpolation error of TETLIN, are shown in Table 3.4.

From Table 3.4 it is clear that the average interpolation error for TETTRI is an order of magnitude less than the average interpolation error of TETLIN. However, the standard deviation is rather large, and the last column shows that there are many samples where the relative interpolation error for TETLIN is less than the interpolation error for TETTRI.

These tests used CFD data on CFD grids. To determine whether similar results can be found with arbitrary hexahedra and “arbitrary” data, we did the
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

<table>
<thead>
<tr>
<th>data field</th>
<th>\text{TetLin} average</th>
<th>\text{TetLin} std. dev.</th>
<th>\text{TetTri} average</th>
<th>\text{TetTri} std. dev.</th>
<th>\text{TetLin} &lt; \text{TetTri} percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs pressure</td>
<td>1.3e-2</td>
<td>8.4e-2</td>
<td>2.0e-3</td>
<td>2.7e-2</td>
<td>6.7%</td>
</tr>
<tr>
<td>bfs velo-x</td>
<td>9.5e-2</td>
<td>2.9e-1</td>
<td>3.7e-3</td>
<td>3.6e-2</td>
<td>6.2%</td>
</tr>
<tr>
<td>bfs velo-y</td>
<td>4.7e-2</td>
<td>2.0e-1</td>
<td>1.9e-3</td>
<td>1.6e-2</td>
<td>3.1%</td>
</tr>
<tr>
<td>bfs velo-z</td>
<td>1.5e-1</td>
<td>3.6e-1</td>
<td>4.3e-3</td>
<td>4.5e-2</td>
<td>1.8%</td>
</tr>
<tr>
<td>fin density</td>
<td>1.8e-3</td>
<td>4.1e-3</td>
<td>4.9e-5</td>
<td>1.4e-4</td>
<td>3.1%</td>
</tr>
<tr>
<td>fin stagnation</td>
<td>1.9e-3</td>
<td>6.0e-3</td>
<td>6.3e-5</td>
<td>4.6e-3</td>
<td>2.8%</td>
</tr>
<tr>
<td>fin moment-x</td>
<td>2.5e-2</td>
<td>6.6e-2</td>
<td>1.1e-3</td>
<td>9.6e-3</td>
<td>2.6%</td>
</tr>
<tr>
<td>fin moment-x</td>
<td>3.3e-2</td>
<td>2.5e-1</td>
<td>2.6e-3</td>
<td>7.2e-2</td>
<td>3.4%</td>
</tr>
<tr>
<td>fin moment-x</td>
<td>3.3e-2</td>
<td>2.5e-1</td>
<td>2.1e-3</td>
<td>7.4e-2</td>
<td>1.3%</td>
</tr>
</tbody>
</table>

Table 3.4: Relative interpolation errors: average values and standard deviation for data of two CFD datasets. The last column shows the fraction of samples at which the relative interpolation error of TetTri is greater than the interpolation error of TetLin.

same interpolation test for a series of arbitrarily deformed cubical cells. Each cell was generated by shifting each of the nodes of a unit cell \([0,0,0]x[1,1,1]\) with a random vector. The x-component \(r_x\) of this random vector \(r\) is determined by

\[
r_x = (\text{rand}(0,1) - 0.5) \times s
\]

with \(\text{rand}(0,1)\) a random generator that returns a random number on the interval \([0,1]\), and \(s\) a scale factor for the random displacement. The other components of \(r\) are generated in the same way. For each node \(n\), at position \((x_n, y_n, z_n)\) three data values are calculated using the following functions:

\[
\begin{align*}
\sin(x_n + y_n + z_n) \\
\exp(x_n + y_n + z_n) \\
x_n + y_n + z_n
\end{align*}
\]

Next, several random positions were chosen in these cells, and values were interpolated in three different ways: TriLin, TetTri and TetLin. The relative errors of TetTri and TetLin, with respect to the value of TriLin were calculated.

In Figure 3.18, we show the averages and standard deviation of the errors for 20 of those cells, where 10 random positions were chosen in each cell. From this figure, it follows, for slightly deformed cells, that the interpolation error of TetTri is an order of magnitude less than the interpolation error of TetLin. (The exception is the third case: as the data varies linearly, TetLin will give exact results.) Also, just as in the CFD-grid tests, the standard deviation is relatively large. Note that TetLin does indeed not depend much on the cell shape, and that for TetTri, both the cell deformation and the data determine the interpolation error, as was to be expected.
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Figure 3.18: Relative interpolation errors for randomly generated cells (left for TetLin, right for TetTri) as a function of the cell random displacement scale factor, for three functions: sin(x_n + y_n + z_n) (upper row), exp(x_n + y_n + z_n) (middle row), and x_n + y_n + z_n (lower row).
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Interpolation is an important element of streamline integration algorithms. Therefore, we also did some experiments to compare (absolute) interpolation errors with streamline integration errors, to find out at what time step size the interpolation errors are approximately as large as the integration errors. For these tests, we use a second order Runge Kutta integration technique, and the same datasets as used for the above described tests. Again, we take some samples in each cell: five samples for the backward facing step and two samples for the fin. At each sample, the data (velocity or momentum) is interpolated, and the absolute interpolation errors are calculated. To calculate integration errors for the same positions, several pairs of streamlines are integrated from these positions. Each pair consists of a streamline of one step with step size \( t \) and a streamline of two steps with step size \( t/2 \). The distance between the end positions of a pair of streamlines is a measure for the integration error of the first streamline. Each next pair of streamlines is traced with half the step size of the previous pair. These traces yield a series of integration errors at different step sizes, which can be compared with the interpolation errors.

The time step of the first trace is chosen such that 6 cells will be traversed in one step, which is achieved by dividing a measure for the cell size (the cell diagonal) by 6 times the average velocity for the cell. This appeared to a suitable initial time step, as with this time step, the initial integration error was always larger than the interpolation error. For each sample position in the dataset, we calculated the integration step size intervals where the integration error is of the same order as the interpolation error, i.e. the integration error at the upper bound is greater than the interpolation error, and the integration error at the lower bound is less than the interpolation error. These results are shown in histograms, in Figure 3.19 for the backward facing step dataset, and in Figure 3.20 for the fin dataset. The average number of steps per cell for which the integration error is of the same order as the interpolation error are shown in Table 3.5. From the histograms and the averages, it follows that generally TETTRI is better than TETLIN. Also, the TETTRI interpolation errors become more significant than the integration errors at approximately 2.5 steps per cell, which thus is an upper bound on the minimum integration step size. This boundary is .75 for TETLIN interpolation.

<table>
<thead>
<tr>
<th>dataset</th>
<th>steps per cell</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TETLIN</td>
</tr>
<tr>
<td>bfs</td>
<td>.72</td>
</tr>
<tr>
<td>fin</td>
<td>.81</td>
</tr>
</tbody>
</table>

Table 3.5: Average number of steps per cell for which the integration error is of the same order as the interpolation error for TETLIN and TETTRI interpolation, in two datasets: backward facing step and fin.
Figure 3.19: Histograms of integration time step intervals that correspond with TetLin (a) and TetTri (b) interpolation errors, for the backward facing step dataset.
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Figure 3.20: Histogram of integration time step intervals that correspond with TetLin (a) and TetTri (b) interpolation errors, for the fin dataset.
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Finally, we did some timing tests, to determine the timing relation between the interpolation techniques. In Table 3.6 we show the results for both a scalar and a vector variable. The timings for the basic steps of these algorithms (actual interpolation, calculation of \( \alpha_t \), and \( \alpha \) from \( \alpha_t \)) are shown in the upper part of the table; these are combined for the interpolation technique in the lower part of the table. The timings were performed on SGI Indy, analogous results where obtained on other machines. The table shows that direct trilinear interpolation is the fastest option, if the fractional offsets are known. Calculation of the exact fractional offsets requires an expensive iteration (see Section 3.3). For the alternative interpolation techniques, it appears that calculation of the fractions \((\alpha_t, \beta_t, \gamma_t)\) inside a tetrahedron is the most time-consuming.

<table>
<thead>
<tr>
<th>operation</th>
<th>scalar</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. trilinear interpolation</td>
<td>2.9</td>
<td>4.4</td>
</tr>
<tr>
<td>2. linear interpolation</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>3. calculation of ( \alpha_t )</td>
<td>9.0</td>
<td>9.0</td>
</tr>
<tr>
<td>4. calculation of ( \alpha ) from ( \alpha_t )</td>
<td>1.3</td>
<td>1.3</td>
</tr>
<tr>
<td>TRILIN (( \alpha ) given: 1.)</td>
<td>2.9</td>
<td>4.4</td>
</tr>
<tr>
<td>TETLIN (2. + 3.)</td>
<td>10.0</td>
<td>11.0</td>
</tr>
<tr>
<td>TETTRI (1. + 3. + 4.)</td>
<td>13.2</td>
<td>14.7</td>
</tr>
</tbody>
</table>

Table 3.6: Timings for interpolation techniques in \( \mu s \) on SGI Indy. The upper part of the table shows timings for components of interpolation technique. The lower part shows total timings for different interpolation techniques.

3.5 Example

Point location and interpolation techniques are applied in several visualization algorithms. Interpolation is used in virtually any visualization algorithm, while point location has applications in streamline integration and ray casting. Streamline integration examples will appear in some of the following chapters, therefore we will give here an example of volume rendering of data on a curvilinear grid, using ray casting.

For our example we use the h-buffer program [83]. This program uses an h-buffer (hybrid buffer) in order to combine volume rendering with rendering of lines and polygons. Volume data is rendered with ray casting techniques, while lines and polygons are scan converted. To speed up the ray casting process, the boundaries of the volume data are projected onto the screen, so that for ray casting the position where the ray enters and leaves the data volume is readily available.

The example uses the result of a fluid flow simulation in a backward facing step geometry. The grid is curvilinear, and consists of 8325 nodes. In our example,
the pressure is visualized as a density field (the cloud). This is combined with two streamlines, and the boundaries of the volume. The streamlines were generated with tetrahedral traversal and trilinear interpolation, for ray casting we used tetrahedral traversal and linear interpolation in tetrahedra. The resulting image is shown in Figure 3.21.

![Image](image)

**Figure 3.21**: Volume rendering of velocity magnitude in backward facing step, combined with two streamlines.

### 3.6 Conclusions

In this chapter, we investigated point location and interpolation techniques for curvilinear grids. Our goal was to develop and compare point location and interpolation techniques that operate in $\mathcal{P}$. Three techniques were described for point location, one iterative search technique: Stencil Walk, and two geometric techniques: tetrahedrization of cells and using bilinear patches as cell boundaries. Of these, Stencil Walk performs superior for global point location, as it is fast in traversing grids over long distances, and as it does not necessarily stop at grid boundaries.

We also described three interpolation techniques: TRI\text{LIN}, TET\text{LIN} and TET\text{TRI}. If the fractional offsets of positions are known, as is the case in the Stencil Walk algorithm, TRI\text{LIN} interpolation should be used. For geometric traversal techniques, TET\text{LIN} and TET\text{TRI} can be used (TRI\text{LIN} can also be used, but if the fractional offsets have to be determined, it is more efficient to use the Stencil Walk algorithm instead of tetrahedrization).

There are two important applications for (local) point location: ray casting and streamline integration. For ray casting, where we have to find the intersections of the ray with the cell boundaries, a geometric technique is necessary.
CHAPTER 3. VISUALIZATION TECHNIQUES ON CURVILINEAR GRIDS

Tetrahedrization is the appropriate technique for ray casting, as it is much faster than the bilinear patch approach. Also, we can use linear interpolation with tetrahedrization, as small interpolation errors will not be noticeable in a volume rendered image.

For the streamline integration, we can combine the interpolation timing data from Table 3.6 with the traversal timings in Figure 3.14–3.16. Then the intersection of the line representing Stencil Walk traversal timings and the line representing tetrahedral traversal will shift to the right. However, the new intersection positions do not differ much from the positions without interpolation. Therefore the conclusions with respect to the traversal algorithms will be the same if we include interpolation: a fast and accurate approach is the combination of tetrahedrization and TETTRI interpolation. A bit faster, and generally less accurate, is tetrahedrization and TETLIN interpolation via tetrahedrization. These techniques can be used if the number of steps per cell is moderate (2.5 or less), as the integration error is greater than or of the same order as the interpolation error. The most accurate, and the slowest, integration technique is the Stencil Walk algorithm, combined with trilinear interpolation. This technique should be used if many steps per cell are taken, as the interpolation mainly determines the error in the streamline tracing.

The relation between the integration and interpolation error can also be interpreted in the following way. If the number of integration steps per cell is very small (less than .75), it does not make sense to use an expensive and accurate interpolation technique, thus tetrahedral traversal with TETLIN interpolation can be used. For many steps per cell (greater than 2.5), the interpolation error becomes more significant than the integration error, thus Stencil Walk with TRI LIN interpolation is the right choice. For a moderate number of steps per cell (between .75 and 2.5) tetrahedral traversal with TETTRI interpolation is a good choice. A summary of these results is shown in Table 3.7.

Finally, from the experiments described, we can conclude that it is not necessary to use computational space for visualization of data on computational grids.

<table>
<thead>
<tr>
<th>steps per cell</th>
<th>technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>few (&lt; .75)</td>
<td>tetrahedral, TETLIN</td>
</tr>
<tr>
<td>several (.75 .. 2.5)</td>
<td>tetrahedral, TETTRI</td>
</tr>
<tr>
<td>many (&gt; 2.5)</td>
<td>Stencil Walk, TRI LIN</td>
</tr>
</tbody>
</table>

Table 3.7: Relation between number of steps per cell and accuracy of interpolation techniques. The integration error (second order Runge Kutta) is of the same order as, or less than, the interpolation error for the given number of steps per cells (average values over two CFD datasets).
Chapter 4

Selective visualization

Small (2D) datasets can be visualized successfully because all data can be displayed simultaneously. The human visual system is very well adapted to processing images that result from such visualizations; the information is extracted from the images by our eyes and brain. This way of visualization is similar to the way graphs are used to analyse a table of numbers. From a simple x-y-graph, we can immediately see the trend of a series of numbers, localize extreme values and detect discontinuities. Likewise, trends, extrema and discontinuities can be detected from a visualization of a small dataset, provided a suitable mapping for the data is chosen.

Unfortunately, datasets often tend to be too large to be projected onto the screen directly: there is too much data to display simultaneously. Solutions to this problem can be found in several directions:

- develop new mappings that allow for even more data to be displayed simultaneously
- develop techniques that select a subset of the data and display just this subset
- develop techniques that extract the information from the data, and present this information instead of the original data

The first approach tries to extend current visualization techniques to larger datasets. As datasets will continue to grow in size and complexity, this approach is bound to fail.

In the second approach, not all data is visualized, but only a subset. From this subset, scientists should be able to extract information. An example has been mentioned in Chapter 2: instead of visualizing an entire 3D scalar field, a cutting plane coloured with a scalar value, moving through the volume, can be visualized. The cutting plane shows only a 2D subset of the data, and from the plane moving
CHAPTER 4. SELECTIVE VISUALIZATION

through the volume, a scientist can mentally reconstruct the data, and extract information.

The third approach is quite different: instead of presenting all data, and letting the scientist process the data visually, this approach aims at extracting the information automatically. Algorithms can be developed that search datasets for information, and display the results of this search. This approach is interesting, but it is generally not simple and not in all cases possible to give a clear description of what is to be considered information. It can only be applied in cases were it is clear what information has to be extracted from the data. Specific, well-tuned algorithms can be developed for such problems. Examples are the visualization of vector field topology [27, 23], and the extraction and visualization of vorticity tubes in turbulent flow fields [81, 3]. Because of the diversity of problems that are solved with visualization, it is impossible to generalize these algorithms to a level where they can be used to search all kinds of information in the data.

Our approach, selective visualization [82, 84], is an attempt to combine the second and third approach. The main idea of our approach is to use a selected part of the data for visualization, where the selected part (selection) is not determined by spatial criteria, but by the data itself. This last statement is important: because the selection is determined by the data, it contains information on the data, that can be used in the visualization mapping to get a better visualization. The selective visualization approach is described in this chapter. In Section 4.1 it is shown how selective visualization can be incorporated in the visualization pipeline, Section 4.2 describes how we define selections and how they can be generated, Section 4.3 describes techniques to process selections and Section 4.4 discusses how visualization mapping can benefit from selections. Section 4.2–4.4 will be accompanied by examples and figures. For clarity, some of these will be in 2D. Of course, the concepts explained extend to (and are also implemented for) the 3D case.

4.1 Selective visualization model

The selective visualization approach consists of three stages: first selections are created, then selections are filtered and finally the selections are used in the visualization mapping stage. This sequence of operations is similar to the standard visualization pipeline, as described in Section 2.1. The standard pipeline consists of a series of transformations that are applied to the data: original data are filtered, filtered data are transformed to visual primitives and visual primitives are transformed to images.

Because of the similarity of both processes, selective visualization can be easily incorporated in the visualization pipeline. The selective visualization process is indeed an extension of the visualization pipeline: an extra data type (selections) is added, and consequently some extra transformations are introduced. These transformations are:
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.1: Selective visualization pipeline, boxes contain operations on data
CHAPTER 4. SELECTIVE VISUALIZATION

- selection creation, which transforms data into selections,
- selection processing, which filters selections,
- selective visualization mapping, which transforms data and selections into visual primitives.

The selective visualization pipeline is show in Figure 4.1. The data preparation stage of the standard visualization pipeline (see Figure 2.1) is subdivided into three transformations: field data preparation (filters field data), selection creation and selection processing. Selective visualization mapping is an extension of visualization mapping. It uses both field data and selection data to generate visual primitives.

4.2 Selection creation

Selection creation is the first stage of selective visualization. The purpose of this stage is to filter out regions in the dataset that contain interesting data, or that can be used to visualize interesting data. The semantics of a selection, or what it means to say that parts of a dataset are interesting, are determined by the application area, the purpose of the research, the phenomena studied and the personal approach of the user. Our problem here is to provide a general technique that enables scientists to specify their areas of interest.

Most visualization systems contain a set of selection techniques. Generally, this set is restricted to spatial selection techniques: the data that are selected satisfy some spatial selection criterion, such as slices of a dataset, cutting planes through a dataset or subvolumes of a dataset. The function of these selection techniques is to reduce the amount of data to be visualized. Application of these techniques is therefore limited to cases where scientists are able to extract the information from the visualization of the selected subset of the data. Visualizations can be enhanced by combining several selections into one image, or by displaying a selection that varies in time. If the scientist knows where to inspect the data, these selection techniques are sufficient.

A more sophisticated selection technique is described by Foley and Lane [20]. Their idea is to use the location of an isosurface to display extra data. In this way two scalar datasets are visualized: one dataset by the isosurface, and the other at the isosurface (e.g. by colouring the isosurface). This technique is an example of a content-based selection technique: the selection (isosurface) is based on the local data value. Similar content-based techniques are available in visualization packages such as AVS [1], where it is possible to colour geometries (visual primitives) such as isosurfaces and streamlines with the value of a scalar field.

Our selection creation approach uses a content-based selection technique, that allows for a general way to select areas in the dataset. The selected areas, called selections, are areas that satisfy some criterion: the selection expression. These
CHAPTER 4. SELECTIVE VISUALIZATION

selection expressions are discussed in the next section (4.2.1), followed by a section on the representation of selections (4.2.2) and a section on the evaluation of selection expressions (4.2.3).

4.2.1 Selection expressions

From the definition of selections, it follows that selection expressions are crucial in selective visualization. Indeed, as they determine the selection, the right choice of expressions determines the success of a visualization. In this section, we address a way to specify selection expressions that can be used to create selections in a general way. Again, we do not address the issue which expressions should be used in specific visualization cases, as this is determined by the application area and the user. Our task is to provide the user with a technique to formalize the criteria of “interestingness”.

To determine whether a certain area belongs to a selection, we need a Boolean selection criterion, i.e. a criterion that yields TRUE or FALSE upon evaluation. Our selection technique therefore consists of a language to specify a Boolean expression, and a mechanism to determine the areas in the dataset that satisfy this selection expression. The language to specify these selection expressions contains many operators and functions, with which one can create virtually any selection that is based on the data values, and also on the grid node positions. In Appendix B a precise definition of the syntax of selection expressions is given, a verbal description follows below. The language uses a C-like syntax [41].

Selection expressions are, as stated before, Boolean expressions, and may contain the following elements:

- Boolean subexpressions, that are combined using the Boolean operators AND (&&), OR (||) and NOT (!),
- elementary Boolean expressions, that consist of a Boolean variable, or a comparison (<, <=, >, >=, ==, !=) of two scalar expressions,
- scalar expressions, that consist of a scalar variable, or an expression or a function yielding a scalar variable (+, -, *, /, ^, sin(), cos(), exp(), log(), len(), ...),
- vector expressions, that consist of a vector variable, or a function yielding a vector variable (cross product, vector addition, vector multiplication, matrix-vector multiplication)
- matrix expressions, that consist of a matrix variable, or a function yielding a matrix variable (matrix transpose, matrix addition, matrix multiplication).

Apart from these Boolean and arithmetic elements, whose evaluation is straightforward, there are other, more complex functions, whose evaluation is not trivial:
- gradient functions, that calculate gradients for scalar and vector variables:
CHAPTER 4. SELECTIVE VISUALIZATION

- gradient of vector or scalar $p$ ($\nabla p$): $\text{grad}(p)$,
- divergence of vector $v$ ($\nabla \cdot v$): $\text{div}(v)$,
- curl of vector $v$ ($\nabla \times v$): $\text{curl}(v)$.

- statistical functions, that calculate minima, maxima, and average ($\text{min}()$, $\text{max}()$, $\text{ave}()$) for the whole dataset, or for a selection of the dataset.

Variables and numerical parameters can be elements of the Boolean expressions:

- data values: by referring to the data name
- grid positions: $xc$, $yc$, and $zc$ as scalar values, or $\text{node}$ as a vector
- grid indices: $i$, $j$ and $k$ as scalar values, or $\text{index}$ as a vector
- numerical parameters that appear in the user interface (and can be changed via the user interface)
- numerical constants: scalars or vectors (between [ ])

Some sample expressions with their meaning are shown in Table 4.1.

<table>
<thead>
<tr>
<th>expression</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P &gt; \text{ave}(P)$</td>
<td>pressure greater than average pressure</td>
</tr>
<tr>
<td>$P &gt; \text{ave}(P) &amp;&amp; \text{len}(V) &gt; 1.$</td>
<td>pressure greater than average pressure and velocity magnitude larger than 1.</td>
</tr>
<tr>
<td>$\text{div}(V) &gt; .5$</td>
<td>divergence of velocity larger than .5</td>
</tr>
<tr>
<td>$\text{len}(\text{node}) &lt; 1.$</td>
<td>distance to origin less than 1.</td>
</tr>
</tbody>
</table>

**Table 4.1**: Sample selection expressions, with $P$ (pressure) and $V$ (velocity) a scalar and vector data field.

The selection expression evaluation mechanism is built in an AVS module (cbs, see Appendix A). It is possible to create new data fields with this AVS module. These data fields contain the result of the evaluation of an expression. In the implementation, expressions are not restricted to Boolean expressions, it is also possible to generate new numerical fields. The following assignment operators can be used to generate new data fields:

- $a=$ generates an Arithmetic data field (numerical), e.g. $\text{newfield} a= \text{len}(V)$ creates a field called $\text{newfield}$ that contains the magnitude of the vectors of a vector field $V$.

- $b=$ generates a Boolean data field, e.g. $\text{newselect} b= S > 0$ creates a selection called $\text{newselect}$ that contains the are where field $S$ is positive.
A third assignment operator is:

- c = generates a Constant, e.g. newconst c = max(S) creates a constant called newconst that contains the maximum value of a scalar data field S.

Finally, there are some special keywords that can be used as the left hand side of an assignment in the cbs-module, that cause the result of an evaluation to be copied to an output port of the module. There is one output port for a Boolean assignment (select.out), one output port for an arithmetic assignment (field.out), and two for constant values (min.out and max.out).

4.2.2 Representation of selection

Until now, selections have been viewed as areas in the dataset that satisfy a selection expression. If we want to apply selection expressions to a dataset, we need a representation for selections: we need a way to store which (spatial) areas in the dataset belong to the selection. This representation should represent the selection accurately, and it should be possible to evaluate selection expressions with this representation.

There are two kinds of representations for areas: continuous and discrete. The continuous one, of course, can be very accurate. One can use geometric modelling techniques to represent selected areas in the dataset. Such techniques allow a wide variety of areas to be represented, as the geometric models can be arbitrarily complex. These continuous representations do have, however, some disadvantages.

The main problem of continuous representations is that data for visualization is often represented on a grid. To obtain a continuous representation of a selection that matches the input data, we need a continuous representation of the input data, and we should evaluate the expression for the continuous data. Of course, we can derive a continuous representation of the input data using some interpolation technique, e.g. by bi- or trilinear interpolation. However, interpolation techniques pose some problems if we want to evaluate selection expressions for a continuous region, and compose a selected area. Let us consider a simple 2D vector field of 2x2 grid nodes, as depicted in Figure 4.2, and let us use bilinear interpolation.

We want to evaluate the following expressions ($V_Y = y$-component of vector, $V = vector$)

$$V_Y > 0$$ - area with upward vector

$$\text{len}(V) > .5$$ - area with large vector magnitude

We determine the areas for which the selection expressions yields TRUE. The evaluation of the first one is simple; the bilinear interpolation of $V_Y$ ($I_{V_Y}$) is larger
than zero (with $\alpha$ fraction along x-axis, and $\beta$ fraction along y-axis):

$$I_{y} > 0 \iff -\alpha(1-\beta) + (1-\alpha)\beta > 0 \iff \alpha > \beta$$

So the boundary of the selected surface is described by the line $\alpha = \beta$.

The second one is less simple:

$$\text{len}(I_{y}) > 0.5 \iff \text{len}(- (1 - \alpha)(1 - \beta) + \alpha \beta, - \alpha(1 - \beta) + (1 - \alpha)\beta) > 0.5 \iff (- (1 - \alpha)(1 - \beta) + \alpha \beta)^2 + (- \alpha(1 - \beta) + (1 - \alpha)\beta)^2 > 0.25 \iff (\alpha - 0.5)^2 + (\beta - 0.5)^2 > 0.125$$

But this can still be solved analytically: it results in a circle with radius $\sqrt{0.125}$ at $(0.5,0.5)$. For more complex expressions with nonlinear operators, the equations cannot be rewritten to such simple forms. In these cases, it would be error-prone and time-consuming to generate an accurate continuous geometric model of the area that satisfies the selection expression. Thus, continuous representations, although accurate, give problems in determining the regions within a cell that satisfy the selection expression.

The alternative is to use a discrete representation, similar to the representation of the data to visualize. If we represent a selection by a Boolean dataset on the grid of the data, we can use an identical representation for selection and input data. This approach does not share the interpolation problems of the continuous representation: the selection data can be generated by the evaluation of the selection expression at each grid node. As all data is known at these nodes, no interpolation is necessary.

The disadvantages of this approach are the limitations of the representation of a selection as a set of grid nodes. It is not possible to represent an arbitrary area with a set of grid nodes: a cutting plane through a dataset might contain not a single grid node, and therefore cannot be represented with this approach. Also, arbitrary positions, such as critical points (points in the vector field where
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.3: Cells are selected if enough corner nodes are selected; gray levels show selected cells at different values for \( n \).

the velocity vanishes) cannot be represented if the critical points do not coincide with grid points.

In our opinion, the disadvantages of the continuous representation are worse than the disadvantages of the discrete representation. Furthermore, both straightforward evaluation and simple representation speak in favour of discrete representation. The accuracy of continuous representation is not important, as the evaluation of complex selection expressions for continuous regions becomes very difficult. Therefore, we represent selections as Boolean datasets on the same grid as used for the data.

In the following, we will use “set of grid nodes” as a synonym for Boolean datasets. Note that we use the term set here in a loose sense, as the set does not only contain a number of scattered positions, but also topological information. The elements in the set are nodes in a grid, and this grid information is maintained in the set.

A disadvantage of a discrete representation of a selection is that it does not allow for continuous selected areas. It is, however, possible to switch to a continuous meaning of selections. Two terms will be used in this sense:

- **selected cell**: a cell where at least \( n \) (with \( n \) some constant) of its nodes are selected, see Figure 4.3,

- **selected region**: a connected set of nodes that are selected (with connectivity either 4- or 8-connectivity for 2D datasets, and 6-, 18- or 26-connectivity for 3D datasets), see Figure 4.4.

Note that these continuous cells and regions are based on sets of grid nodes, and determined after the evaluation of the selection expression. Even if all its nodes satisfy the expressions, the selection expression may not be satisfied throughout a selected cell. An example is shown in Figure 4.2. All corner nodes a the cell
satisfy the expression \( \text{len}(v) > .5 \), but with bilinear interpolation, the vector length at the cell centre will be 0.

### 4.2.3 Evaluation of expressions

Expression evaluation is the process which transforms a selection expression into a selection. Its input is a data set and a selection expression, its output is a selection. Figure 4.5 shows how this process is implemented: first a tree is built that represents the expression, then the types of the operand are checked, then constant subexpressions are evaluated, and finally the expression is evaluated for each grid node, yielding a selection.

The tree building is implemented with lex [46] and yacc [35]. Lex is used to cut the selection expression into a series of tokens, and yacc is used to transform this series of tokens into a tree that represents the expression. Figure 4.6 shows the tree built from the expression

\[
\text{select}_{\text{out}} \text{ b=} \text{len}(\text{velo}) > .5 \&\& \text{pres} < .5 \times \text{max(} \text{pres} \text{)} .
\]

Syntactic checking of the expression is performed in this step.

Next, it is checked whether this tree can be evaluated. Two tests are performed:

- do the identifiers used in the expression exist? (i.e. do the names of the
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.5: Steps in transformation of a selection expression into a selection.

Figure 4.6: Tree built from the selection expression `select_out b= len(velo) > .5 && pres < .5*max(pres)`
identifiers correspond to variables in the data set or previously calculated variables?)

- are the types of the operands correct? (some operators or functions take scalar arguments, some take vector arguments)

Type checking is the only semantic check of the tree. After these checks, the tree is ready to be evaluated.

The nodes that belong to the selection are determined by the evaluation of the tree for each node. However, repeated evaluation of a tree as shown in Figure 4.6 would be rather inefficient, as it would require a repeated calculation of the maximum pressure, while the maximum pressure will evaluate to the same value for each node. Therefore, evaluation of the tree for all nodes is preceded by an evaluation of constants. All constant subtrees (such as 3* .5) and constant-valued subtrees (such as max (pres)) are evaluated, and the corresponding subtree is replaced by a constant subtree (node). In Figure 4.7, the result of constant evaluation for the tree of Figure 4.6 is shown. It is assumed that the maximum value for pressure is equal to 5.

Maxima, minima and averages are calculated in this constant evaluation step. Global maxima and minima are determined by scanning the whole data set and storing the maximum (minimum) found. Averages are calculated by averaging over the number of nodes, the size of the cells that surround a grid node is not taken into account (a better technique to calculate these values is discussed in Section 5.2.1). If grid spacing varies much, this average value might differ from the real average value.

After this constant evaluation, the tree is evaluated for each node to create a selection. If the right-hand-side of the expression evaluates to TRUE for a grid node, then the node belongs to the selection, else the node does not belong to the selection. Evaluation of the tree for a node is straightforward for the Boolean
and arithmetic operands and functions. The evaluation of the gradient functions is not straightforward, and is discussed below.

The way gradients are calculated depends on the grid type on which the data is defined. For Cartesian grids we use central difference techniques to calculate partial derivatives. With \( h_x \) the distance between nodes in x-direction, and \( s[i,j] \) the value of a quantity \( s \) at node \([i,j]\), the following forward differencing formula is used to calculate the partial derivative of \( s \) at node \([i,j]\) with respect to \( x \):

\[
\frac{\partial}{\partial x} s[i,j] = \frac{s[i + 1,j] - s[i - 1,j]}{2h_x}
\]  

(4.1)

Similar formulas are used for derivatives with respect to \( y \), and for the 3D case. With these partial derivatives one can calculate gradient, divergence and curl in the standard way.

Figure 4.8: Neighbours of a grid node that are used for the calculation of partial derivatives of a quantity at \( p \).

Calculation of partial derivatives for a curvilinear grid is more complex. We implemented a technique described by Dekker [12]. It uses the second order Taylor approximations of a quantity \( s \) about node \( p \), with respect to its neighbouring nodes \( n_i \). For each of the neighbouring nodes \( n_i \) (see Figure 4.8), the following equation holds:

\[
s(n_i) \approx s(p) + ((n_i - p) \cdot \nabla)s(p) + ((n_i - p) \cdot \nabla)^2 s(p)
\]  

(4.2)

With \((dx_i, dy_i)\) the vector from \( p \) to \( n_i \), and \( s_x, s_y, s_{xx}, s_{xy} \) and \( s_{yy} \) the partial derivatives of \( s \), and neglecting the error in the Taylor-approximation, equation 4.2 leads to the following system of equations:

\[
\begin{pmatrix}
s(n_1) - s(p) \\
\vdots \\
s(n_8) - s(p)
\end{pmatrix}
= \begin{pmatrix}
dx_1 & dy_1 & dx_1^2 & dx_1 dy_1 & dy_1^2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
dx_8 & dy_8 & dx_8^2 & dx_8 dy_8 & dy_8^2
\end{pmatrix}
\begin{pmatrix}
s_x(p) \\
s_y(p) \\
s_{xx}(p) \\
s_{xy}(p) \\
s_{yy}(p)
\end{pmatrix}
\]  

(4.3)

where \( s(n_i), s(p), dx_i \) and \( dy_i \) are known, and the partial derivatives unknown. This system is solved with a pseudo inverse matrix technique [5]. For nodes at
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.9: Nodes that are used for partial derivative calculation if node \( p \) is at the grid boundary (left) or at a corner of the grid (right).

the interior, we use the surrounding neighbouring nodes, for boundary nodes we use additional interior nodes (see Figure 4.9).

The same technique can be applied for 3D grids. In that case, there are 9 unknowns \( (s_x, s_y, s_z, s_{xz}, s_{xy}, s_{yz}, s_{xZ}, s_{yZ}, s_{zZ}) \). We use all 26 neighbours of a grid position, so we get 26 equations. The system is solved with the same pseudo inverse matrix technique. Note that we have to invert the matrix with \( dx_i \) and \( dy_i \) only once in order to calculate partial derivatives at a node, as the distances do not change for other data. So, for the calculation of the divergence of a 3D vector field, we need one matrix inversion (to invert the distances matrix) and three matrix-vector multiplications (one for each vector element) per node.

4.3 Selection processing

Areas in the dataset can be selected, based on the local value of variables in the dataset with the techniques described in the previous section. The result of this selection process is a Boolean dataset.

The selection processing stage contains additional techniques to manipulate selections. As selection might not be well suited to use directly in the mapping stage, one needs techniques that allow for enhancing the selection. Examples of such enhancements are the removal of isolated selected nodes, or the grouping of connected selected grid nodes. These selection processing techniques have a purpose that is similar to the purpose of many image processing techniques: enhancement of images. As a 2D selection, being a structured 2D Boolean dataset, is very similar to a 2D black-and-white image, we use some standard image processing techniques (see e.g. [72]) for processing of selections.

Five techniques have been implemented to manipulate selections: clustering, erosion, dilation, opening and closing. Clustering is the most important selection processing technique. It groups neighbouring selected nodes into regions, based
on some connectivity criterion (see Figure 4.4). Each group of selected nodes (cluster) gets a label, that can be used to select one or more specific clusters in the visualization mapping stage.

_Erosion and dilation_ are techniques that change selections (in contrast to a clustering, that only assigns labels to clusters of nodes). Erosion "erodes" a selected region at the boundary of that region, dilation extends a selected region at the boundary of a that region. Both techniques apply a mask to each node of the selection. In case of erosion, a selected node is set to not-selected if at least one node under the mask is not a selected node. In case of dilation, a not-selected node is set to selected if at least one node under the mask is a selected node. Examples are shown in Figure 4.10. In our implementation, we use a 4- or 8-connected neighbourhood of a node as a 2D mask, and a 6-, 18- or 26-connected neighbourhood of a node as a 3D mask.

![Diagram of selected node and mask](image)

**Figure 4.10:** Erosion and dilation with a 4-connected mask: a) original selection; b) eroded selection; c) dilated selection

Opening is the application of an erosion, followed by a dilation. The result of an opening is that small regions (isolated selected nodes) disappear, that regions connected by a narrow link are separated, and that small details at the boundary of a region disappear. These effects are illustrated in Figure 4.11.

_Closing_ is the opposite of opening, it is the application of a dilation, followed by an erosion. The results of a closing operation are therefore: regions close to each other are connected, and holes in regions are filled (see Figure 4.12).
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.11: Effects of an opening: isolated nodes disappear, connected regions are separated and details at the selection boundary disappear; a) original selection, b) eroded selection, c) opened selection (dilation after erosion).

Figure 4.12: Effects of a closing operation: nearby regions get connected and holes get filled; a) original selection, b) dilated selection, c) closed selection (erosion after dilation).
CHAPTER 4. SELECTIVE VISUALIZATION

The techniques described can be applied to both 2D and 3D selections. There is, however, one problem with the application of such techniques: in general, the pixels in an image are of the same size, and are equally spaced, while curvilinear grids have a variable grid spacing and consequently a varying cell size. Of course, we can disregard the grid node positions and apply the techniques in index space (and this is how we do apply the techniques), but we cannot do this without stating why we are allowed to disregard the grid positions.

The consequence of disregarding the grid positions is that the physical distance between neighbouring grid nodes is not accounted for. For the clustering, this means that two selected nodes that are very close together, yet separated by an unselected node, are not joined, while two nodes that are farther apart, yet not separated, are joined. For erosion (or dilation) this means that the measure of shrinking (or expanding) varies over the grid, and depends on the distance between nodes in physical space.

Despite these consequences, these techniques still can be applied successfully to curvilinear grids, for the following reasons:

- normally the grids are designed for specific simulation problems; the grid spacing is dense where one expects the complexity of the data to be high. It is reasonable to use the same resolution for selection processing techniques.

- these techniques, though necessary, are not critical in their result. They are used to enhance a selection, and not to calculate exact results (as is the case in the visualization mapping algorithms).

- for the opening and closing techniques, there is an additional reason: the sequential application of the complementary erosion and dilation operations greatly reduces the effects of varying distances in physical space. The net effect of filling holes, or separating clusters, remains.

If we want to extract additional information from the selection (such as in iconic visualization), these processing techniques should be applied with some care, and the varying grid spacing should be accounted for.

4.4 Selective visualization mapping

Selective visualization mapping is the stage where raw field data and selections are combined to create visual primitives. The difference with standard visualization mapping is that selective visualization mapping uses selections and selected data (data at selected nodes) in the mapping process. Selections specify the nodes that are considered to contain interesting data, hence selective visualization mappings can generate better visualizations than standard visualization mappings.

Selections play different roles in the mapping stage. Below, we discuss three uses of selections: direct visualization, clipping and parameter extraction. These techniques combine well with standard visualization techniques. We will show
CHAPTER 4. SELECTIVE VISUALIZATION

how some important visualization mappings can be extended to be applied to selections. The techniques have been implemented as AVS-modules. These modules are described in Appendix A, information on the algorithms implemented is included below.

Each technique will be illustrated with a 2D and a 3D example. The selection used for the 2D example is shown in Figure 4.13, in some examples the selected nodes are shown with black dots. The 3D example uses a dataset with a fluid flow in a backward facing step geometry. The selection expression used for example figures in 3D is:

\[
\text{select_out b = } i \neq 0 \&\& i \neq \max(i) \&\& \\
  j \neq 0 \&\& j \neq \max(j) \&\& \\
  k \neq 0 \&\& k \neq \max(k) \&\& \\
  \text{len(bfs_velo)} < 0.035
\]

being all interior nodes where the velocity magnitude is less than .035.

4.4.1 Direct visualization of selections

First of all, selections can be visualized directly. The selection, a set of grid nodes, contains information even without additional data. Visualization of this selection is useful if the selection represents some phenomenon that has to be visualized, for example structures in turbulent flow, as described by Hunt et al. [34]. Hunt et al. give expressions to classify zones in a turbulent flow field. These expressions can be transformed to selection expressions, after which the zones can be shown by direct visualization of the selection.

Another application is the visualization of a selection as an intermediate step in the visualization process. In the process of creating a selection, visualization provides information on the selection result. This information can be used to tune a selection expression, by adjusting threshold values or adding new expressions.

Visualization of a selection is straightforward, it amounts to displaying a set of grid nodes. This can be achieved by displaying a small object, such as a cross-markers or a dot, at each grid node, see Figure 4.13. Another way is to visualize selections as regions, by generating a surface representation of the selection boundary, see Figure 4.14. Assuming that selected nodes have a value of '1', and not-selected nodes have a value of '0', a bounding surface of the selected areas can be constructed as an isosurface for .5. In this way, we can use an isosurface technique for the visualization of a selection.

4.4.2 Clipping with selections

Clipping is another function of selections in the visualization mapping: instead of displaying all data, only those data (or visual primitives) that are inside a selection are displayed. Clipping with selections solves two problems that frequently occur in visualization of 3D datasets by displaying all data:
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.13: Direct visualization of a selection with objects: cross-markers at selected nodes; a) 2D cross-markers; b) 3D cross-markers.

Figure 4.14: Direct visualization of a selection with a surface: a) 2D example: line around a selected region; b) 3D example.
CHAPTER 4. SELECTIVE VISUALIZATION

- redundant or not interesting data can be excluded from the visualization; if we are interested in a velocity field at positions where the pressure is high, we just clip the visual primitives that visualize the velocity field against the selection that contains nodes with high pressure values.

- visual primitives are less likely to occlude each other; for example, if we visualize a set of concentric isosurfaces, the outer isosurface will conceal all inner isosurfaces. If we clip away parts of the outer surfaces, the inner isosurfaces can be inspected.

Another advantage of clipping is that additional data can be visualized. Because clipping results in images with fewer visual primitives, space is available to visualize additional data.

The clipping function of selections can be implemented in the visualization mapping stage or in the rendering stage. In the latter case, all visual primitives must be generated in the mapping stage, even primitives that will not be displayed because they are outside the selection. Of course, this approach is rather inefficient, as more visual primitives than necessary will be generated. A more efficient approach is to decide in the mapping stage which visual primitives must be generated, and which can be skipped. We have followed this approach, and implemented the clipping function in the visualization mapping stage.

For scalar field rendering, we implemented an isosurface algorithm that clips the parts of the isosurface outside a selection (see Figure 4.15). The isosurface algorithm subdivides a hexahedron in five tetrahedra, as described in Section 3.3.3,

![clipped isosurfaces at cell with a selected node](image)

(a)

![clipped isosurfaces at cell with a selected node](image)

(b)

**Figure 4.15:** Isosurface clipping: a) 2D example: isoline clipping against a selection; b) 3D example: three clipped isosurfaces (for pressure equals -1, -.5 and 0.)
Figure 4.16: Isosurface algorithm with tetrahedrization; a) tetrahedrization; b) tetrahedron, nodes above isovalue are black, nodes below isovalue are white; c) isovalue is interpolated along edge, yielding intersection of edge with isosurface; d) triangulation of isosurface using interpolated positions on edges.

see also Figure 4.16. Then, in each tetrahedron, an isosurface is constructed if there is at least one vertex of the tetrahedron with a value above, and at least one vertex with a value below the isovalue. The vertices of the isosurface are determined by interpolation along the tetrahedron edges, the configuration of the triangulated isosurface is determined by the index of the tetrahedron, in a similar way as in the Marching Cubes algorithm [48]. However, as a tetrahedron has only 4 vertices, there are only 16 different indices. Because of this limited number of possible configurations, we did not use any symmetry information to reduce the number of configurations, as done in the Marching Cubes algorithms. This algorithm generates approximately twice as many triangles as the Marching Cubes algorithm, but does not share the inconsistency problems of the Marching Cubes algorithm. Clipping is performed by adding a test whether the cell to be processed is a selected cell.

Another clipped scalar field rendering can be obtained by colouring a selection visualization with the value of a scalar variable, as shown in Figure 4.17. The scalar field visualization is clipped against the selection, so that the visualization only contains the selection. This application is an illustrative example of the advantages of clipping. If a scalar variable is displayed at all nodes with this technique, very little information can deduced from the visualization. However, if only selected regions are visualized with this technique (see e.g. the pictures in Section 6.2), the visualization is very informative.

Hedgehogs can be clipped in a similar way, by displaying arrows only at selected grid nodes, see Figure 4.18. This has been implemented by sampling a selection. A number of sample positions within a selection (all selected nodes, or a fraction of all selected nodes) is chosen from a selection, and can be used as positions to display arrows.
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.17: Scalar field visualization clipping: coloured cross-markers at selected nodes; a) 2D example; b) 3D example.

Figure 4.18: Hedgehog clipping: arrows at all selected nodes; a) 2D example; b) 3D example.
CHAPTER 4. SELECTIVE VISUALIZATION

Streamline clipping is similar to isosurface clipping. A standard streamline algorithm is used, combined with a test to decide whether streamline integration should be terminated. As soon as a selection is left, i.e. if the streamline enters a cell that is not selected, the streamline is not integrated further, see Figure 4.19. This can be combined with parameter extraction for finding starting position for streamline integration inside a selection, as described in the next section.

![Streamline clipping: streamlines are integrated only within a selection; a) 2D example, b) 3D example.](image)

Figure 4.19: Streamline clipping: streamlines are integrated only within a selection; a) 2D example, b) 3D example.

### 4.4.3 Parameter extraction using selections

Algorithms for visualization mappings generally need certain parameters. Some parameters affect the algorithm itself, while other parameters affect the appearance of the visual primitives. Consider a streamline visualization, which maps a velocity field to a number of streamlines. Parameters of the first type are the integration method, and the integration time step; parameters of the second type are the number of streamlines, and the starting positions of streamlines.

One of the problems in visualization is a good choice for these parameters. Sometimes, the values for these parameters are known in advance; if we want to know where some scalar data variable exceeds a threshold, we just draw an isosurface at that threshold. However, often the right values for these parameters are determined by trial and error: various positions for slicers, or streamline integration starting positions are tried until the visualization gives the information desired.

An alternative to this trial and error process is to extract suitable parameters for visualization directly from the data. Starting positions for streamline integration can be chosen from regions with interesting flow patterns (much like Helman
and Hesselink [28] start their streamline integration from critical points), and slices can be put at positions with maximum data values. The advantages of such automated approaches are clear: the visualization can be reproduced easily, and a direct search for the information in the data is supported.

In our selective visualization system, we implemented the following techniques to extract parameter values for visualization algorithms:

- calculation of minimum, maximum and average values over a selected region, to be used in determining a threshold value in e.g. an isosurface or colour map scaling algorithm,

- choosing positions from a selected region to start streamline integration,

- calculating centroids of a selected region, to position slices.

These techniques reflect the two parameters that are mostly needed in visualization algorithms: threshold values and positions. Other, more sophisticated ways to extract visualization parameters from selections, and how to use them in visualization algorithms are described in Chapter 5.

4.5 Example: candidate cells for critical points

In the previous sections, selective visualization techniques have been described, and for each technique we gave a simple example. In this section we give a more elaborate example of selective visualization, in which techniques from all stages are used. The purpose of this example is to show that:

- selective visualization techniques can be used to create complex content-based selections

- selection processing techniques can be applied successfully on curvilinear grids

More examples can be found in Chapter 6.

It has been argued that the selective visualization approach, as described in this chapter, can not be used to visualize flow field topology. The main reason is that critical points can not be selected, as they generally do not coincide with grid points. In this example, we show that the first step in critical point location: searching of cells that might contain critical points, can be performed with the techniques described.

Recall that critical points are points where the velocity vanishes. Generally, critical points are located by looking for candidate cells, and then starting a Newton-Raphson iteration in that cell to find a critical point. A necessary condition for a cell to be a candidate is that all velocity components have a sign change in the cell. If this condition is satisfied, velocity in the cell determined by trilinear interpolation may be a null vector at certain points.
CHAPTER 4. SELECTIVE VISUALIZATION

How can these candidate cells be identified using selective visualization techniques? First, we reformulate the problem, and state that we want to select nodes, that belong to a candidate cell (as the techniques are node based, not cell based). Then, we split the problem into three subproblems: determine the nodes that belong to candidate cells with respect to the x-, y- and z-components of the velocity.

Finding nodes of candidate cells for the x-component of the velocity amounts to finding all nodes that belong to cells with a positive and negative velocity x-component. The nodes with a positive x-component can be selected easily: select.out b = veloX <= 0, nodes with a negative x-component are selected in a similar way. From all nodes with a positive x-component we can get all nodes that belong to a cell with a positive x-component, by dilating the selection with positive x-components (see e.g. Figure 4.10). The same can be done for the nodes with a negative x-component. The intersection (&&) of these selections yields all candidate nodes that belong to cells where the x-component might become zero. If these techniques are also applied for the y- and z-component, we can get all nodes of the candidate cells by combining the candidate nodes for each velocity component. Each cell of which all corner nodes are selected is a candidate cell for critical points.

The selection expressions used to find these candidate cells are shown in Figure 4.20, the AVS-network used is shown in Figure 4.21. The expressions perform the following:

- nodes where velocity has a negative x-component (bfs.veloX) are selected (line 1),
- the result of this selection is fed into the dilation module (lines 2–4)
- the result of the dilation contains all nodes that belong to a cell of which at least one node has a negative velocity x-component (line 5)
- the same is done for positive velocity x-component (lines 7–11)
- these selections are combined, yielding all nodes that belong to a candidate cell, with respect to the x-component of the velocity (line 13)
- the same is done for the y-component (lines 16–25) and the z-component (lines 28–36)
- the nodes of the candidate cells are determined by combining xcandidate, ycandidate and zcandidate (line 39)
- these nodes are output (line 40)

Note that we do not include nodes where velocity is equal to zero (being all boundary nodes, except the inflow and outflow boundaries). These nodes would
CHAPTER 4. SELECTIVE VISUALIZATION

1 xneg b= bfs_veloX < 0 ; nodes with negative x-velocity
2 select_out b= xneg ; put these nodes at output port
3
4 wait ; create dilated version of xneg
5 xneg_dil b= cluster ; cells with a negative x-velocity
6
7 xpos b= bfs_veloX > 0 ; nodes with positive x-velocity
8 select_out b= xpos ; put these nodes at output port
9
10 wait ; create dilated version of xpos
11 xpos_dil b= cluster ; cells with a positive x-velocity
12
13 xcandidate b= xpos_dil && xneg_dil ; nodes from cells with
14 ; a negative and positive x-velocity
15
16 ; similar for y-component
17 yneg b= bfs_veloY < 0
18 select_out b= yneg
19 wait
20 yneg_dil b= cluster
21 ypos b= bfs_veloY > 0
22 select_out b= ypos
23 wait
24 ypos_dil b= cluster
25 ycandidate b= ypos_dil && yneg_dil
26
27 ; similar for z-component
28 zneg b= bfs_veloZ < 0
29 select_out b= zneg
30 wait
31 zneg_dil b= cluster
32 zpos b= bfs_veloZ > 0
33 select_out b= zpos
34 wait
35 zpos_dil b= cluster
36 zcandidate b= zpos_dil && zneg_dil
37
38 ; candidate cells: nodes with x- y- and z-candidate
39 candidate b= xcandidate && ycandidate && zcandidate
40 select_out b= candidate ; put candidate cells at output port

Figure 4.20: Selection expressions for finding candidate cells for critical points.
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.21: Network for finding candidate cells for critical points, visualizing these cells and generating streamlines through these regions.

<table>
<thead>
<tr>
<th>module</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_field</td>
<td>read data</td>
</tr>
<tr>
<td>extract vector</td>
<td>get vector field from data</td>
</tr>
<tr>
<td>volume bounds</td>
<td>generate volume boundaries</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 1–4</td>
</tr>
<tr>
<td>ndip.dilate.erode</td>
<td>dilate xneg</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 5–10</td>
</tr>
<tr>
<td>ndip.dilate.erode</td>
<td>dilate xpos</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 11–19</td>
</tr>
<tr>
<td>ndip.dilate.erode</td>
<td>dilate yneg</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 20–23</td>
</tr>
<tr>
<td>ndip.dilate.erode</td>
<td>dilate ypos</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 24–30</td>
</tr>
<tr>
<td>ndip.dilate.erode</td>
<td>dilate zneg</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 31–34</td>
</tr>
<tr>
<td>ndip.dilate.erode</td>
<td>dilate zpos</td>
</tr>
<tr>
<td>cbs-all</td>
<td>evaluate line 35–40</td>
</tr>
<tr>
<td>select_axis_col</td>
<td>visualize selected nodes</td>
</tr>
<tr>
<td>cb_stream_sw</td>
<td>generate streamlines</td>
</tr>
<tr>
<td>geometry_viewer</td>
<td>render image</td>
</tr>
</tbody>
</table>

Table 4.2: Firing order of modules in AVS network from Figure 4.21, line numbers refer to Figure 4.20.
CHAPTER 4. SELECTIVE VISUALIZATION

Figure 4.22: Nodes that belong to cells that might contain critical points.

Figure 4.23: Streamlines through the selected region from Figure 4.22.
be included if we would replace in Figure 4.20 all occurrences of '<' with '<=', and all occurrences of '>' with '>='.

The expressions from Figure 4.20 are fed into the selection module cbs-all by reading them from a file (in fact, the Figure 4.20 is the file that is used for this). Table 4.2 shows the order in which the modules from the network are executed.

The results of the visualization are shown in Figure 4.22, which contains a visualization of the selected nodes, and Figure 4.23, which displays streamlines through these selected nodes.

### 4.6 Conclusions

In this chapter, the selective visualization approach has been described. We have shown how this approach fits in the visualization pipeline model, and which techniques can be used for selective visualization. Selections have been defined as sets of grid nodes. The critical point candidate cell application has shown that it is possible to use selection techniques to create complex selections.

The selective visualization approach is based on the assumption that not all data in a dataset contribute equally to the information in the dataset. The selection process aims at filtering out the data that contain most information. Selection expressions play a key role in this filtering process.

Indeed, the applicability of selective visualization depends on the possibilities to transform the research question about a dataset to a suitable selection expression, combined with suitable selective visualization mappings. It is our experience that this transformation is not difficult if the research question about the dataset is clearly formulated. Examples of such transformations can be found in the case studies in Chapter 6.
Chapter 5

Iconic visualization

In the previous chapter, we described techniques for selective visualization. Selection expressions were used to select regions in datasets. These regions were visualized with standard visualization techniques that are extended to incorporate selections. In this chapter we introduce another way for visualization using selections: iconic visualization. The goal of iconic visualization is to get a 'summarized' visualization, to visualize essential elements of a dataset in a simple, clear and understandable way. This iconic visualization consists of two stages: a stage in which characteristic values for a selected region of a dataset (attributes) are calculated, and a stage in which the attributes are mapped onto the parameters (degrees of freedom) of parametric iconic objects. Examples of such iconic objects (the 3D equivalent of a 2D Chernoff face [10]) are shown in Figure 5.1. This icon has 20 degrees of freedom.

![Figure 5.1: Three 3D Chernoff faces. (See also Colour plate 6 on page 165.)](image)

A process model for iconic visualization is described in Section 5.1. Then, in Section 5.2, we focus on attributes, their calculation and transformation. Icons, icon design and iconic mapping are discussed in Section 5.3. This chapter is concluded with a some examples (Section 5.4) and conclusions (Section 5.5).

The iconic visualization techniques were intended for use with selection tech-
CHAPTER 5. ICONIC VISUALIZATION

niques. However, while working on this subject, it appeared that the concepts of iconic visualization are applicable to a wider range of visualization techniques. Therefore, we will also give some examples of iconic visualization techniques in a wider context (not just in relation to selective visualization).

5.1 Iconic visualization model

In Section 4.1 we described a model for selective visualization. This model can be extended to incorporate iconic visualization. The first step in the iconic visualization process is to find regions in the dataset that need to be visualized, i.e. data that are important or relevant in some respect. After this data selection step, derived data values for these regions are calculated. This results in a set of attribute values: the attribute set. Each of these values is an attribute of the selected data. In the iconic visualization mapping stage, these attributes are mapped onto the degrees of freedom of an icon. This process is depicted in Figure 5.2.

The process of selection and attribute calculation is a process of abstraction: a region in a dataset is viewed and treated as an entity, and is visualized at a higher level. This abstraction process slightly changes the meaning of a selection: in the previous chapter a selection was viewed as a set of nodes that contains interesting data, while here a selection (or a cluster in a selection) relates to a “feature” ("anything contained in a data set that might be of interest for interpretation", according to Hesselink, Post and Van Wijk [70]). We also do not reserve the term feature for specific physical phenomena in the data, such as vortices, shock waves and turbulent structures in CFD data, but we use it in a wider context: the term feature applies to any region for which an abstract representation and visualization is worthwhile.

Selection of relevant data is a necessary preparation step for iconic visualization. Data can be selected with the techniques described in the previous chapter. The attributes for selected data can calculated done in various ways, depending on what is to be considered a relevant attribute. A generic way to calculate a range of attributes is to calculate volume integrals over selected data. This is described in the next section.

In the iconic mapping stage, the attributes are mapped onto the degrees of freedom of icons. The purpose of this mapping is to visualize selected data by geometric objects, that display the characteristics of a selection in a clear and understandable way. There should be some similarity relation between the selections and their iconic representations. A limited set of icons does not seem appropriate for this visualization approach, therefore we developed an icon modelling language, with which icons can be designed and a binding can be established between the selections attributes and the degrees of freedom of the icons. In Section 5.3, we describe this language, and show some examples of icons and iconic mapping.

One important consequence of the iconic visualization process is a vast data
CHAPTER 5. ICONIC VISUALIZATION

Figure 5.2: Iconic visualization pipeline.
CHAPTER 5. ICONIC VISUALIZATION

reduction. The original data field is replaced by a usually small number of attribute sets, that are visualized by simple geometric objects. The reduction is caused by the data selection and abstraction, and such reduction is necessary for understanding the information inherent in very large data sets.

The size of the intermediate attribute data set is very small, and this provides an excellent opportunity for distributed processing. The computationally intensive first part of a numerical simulation and visualization can be executed on a remote high performance computer, and the results can easily be transferred to the visualization workstation by a low-bandwidth link.

Note that this iconic visualization concept can be generalized easily. Attribute sets can be calculated with other algorithms as well, as is discussed by Post et al. [64]. The same iconic visualization approach can be applied in each case where some preprocessing technique calculates an attribute set that represents important features in a dataset.

5.2 Attributes

Attributes play a key role in iconic visualization: they characterize a selection, and are used in the iconic mapping stage. We call a group of attributes that characterizes one selection an attribute set. An attribute set of a selection characterizes aspects of the selection, and as such is an abstract representation of the selection.

Here, and in the remainder of this chapter, we will use the term selection for a single cluster of a selection. Although the techniques presented do not necessarily require that a selection is a coherent region, it is reasonable to assume that they are. (Techniques for clustering where described in the previous chapter.)

In this section, we describe a generic technique for the calculation of attribute sets for selections: volume integrals. Volume integrals can be used to calculate values such as volume, mass and averages for selections. We also give an example of how attribute sets can be generated with a streamline tracing algorithm. In this case, the attributes do not represent a selection, but a series of 3D positions. Hence, they are also treated differently in the iconic mapping stage. We also describe techniques to transform attribute sets.

5.2.1 Volume integrals for attribute calculation

Selections denote regions in a dataset. Several characteristic values for these regions can be obtained by integration over this region. If \( S \) is a region that belongs to a selection, \( \mathbf{x} \) a position vector, \( \rho \) a density field and \( \mathbf{v} \) a vector field and \( \mathbf{v}^T \) the transpose of \( \mathbf{v} \), then the following quantities can be calculated with volume integrals:
volume of $S$ ($V_S$) \[ \int_S dS \]

centre of $S$ \[ \frac{1}{V_S} \int_S x \, dS \]

second moments of $S$ \[ \left( \frac{1}{V_S} \int_S xx^T \, dS - \frac{1}{V_S} \int_S x \, dS \frac{1}{V_S} \int_S x^T \, dS \right) \]

mass of $S$ \[ \int_S \rho \, dS \]

centre of gravity of $S$ \[ \frac{1}{V_S} \int_S \rho \, x \, dS \]

average $v$ of $S$ \[ \frac{1}{V_S} \int_S v \, dS \]

var./cov. matrix for $v$ of $S$ \[ \left( \frac{1}{V_S} \int_S vv^T \, dS - \frac{1}{V_S} \int_S v \, dS \frac{1}{V_S} \int_S v^T \, dS \right) \]

Thus volume integrals offer the possibility to calculate many different aggregate data values for selections.

We use quadrature rules to determine such volume integrals over selections. The integral of a function over a domain is calculated by adding samples (abscissae) of the function over the domain. For a one-dimensional space, the integral for a function $f(x)$ over a domain $D$ is approximated using:

\[ \int_D f(x) \, dx \approx \sum_{i=1}^k W_i f(x_i) \quad (5.1) \]

with $W_i$ the weight factor for position $x_i$, and the $x_1 \ldots x_k$ samples from $D$. The number of samples, the sample positions and the values for $W_i$ are determined by the type of quadrature. Table 5.1 shows the trapezium rule and a two points Gauss-Legendre quadrature rule for domain $[-1, 1]$. Gauss-Legendre quadrature rules integrate polynomials of degree $(2n - 1)$ exactly with $n$ sample points.

This one-dimensional integration process can be extended to volumes (cells of a grid), by calculating the function to be integrated at $n \times n \times n$ sample points, and

<table>
<thead>
<tr>
<th>Method</th>
<th>Abscissae</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trapezoidal</td>
<td>$-1, 1$</td>
<td>$1, 1$</td>
</tr>
<tr>
<td>Gauss-Legendre, two points</td>
<td>$-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$</td>
<td>$1, 1$</td>
</tr>
</tbody>
</table>

| Table 5.1: Quadratures rules. |
summing over the volume. Assuming a cubical cell $C$ of $[-1,1] \times [-1,1] \times [-1,1]$ in 3D space, the volume integral of $f(x)$ over $C$ is calculated with:

$$
\int \int \int_C f(x) \, dx \, dy \, dz = \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} f(x) \, dr \, ds \, dt
$$

$$
\approx \sum_{i,j,k=1}^{n} W_i W_j W_k f(x_i, s_j, t_k) \tag{5.2}
$$

For a curvilinear grid, we transform hexahedral cells to a unit cubical cell, as used in Equation 5.2, see Figure 5.3. This is done by defining a transformation function $h(r, s, t)$, with $r, s, t \in [-1,1]$, that maps a position from the unit cubical cell to a position in the hexahedron. With $|J(r, s, t)|$ the determinant of the Jacobian of $h(r, s, t)$, Equation 5.2 can be written as:

$$
\int \int \int_C f(x) \, dx \, dy \, dz = \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} f(h(r, s, t)) \, |J(r, s, t)| \, dr \, ds \, dt
$$

$$
\approx \sum_{i,j,k=1}^{n} W_i W_j W_k f(h(r_i, s_j, t_k)) \, |J(r_i, s_j, t_k)| \tag{5.3}
$$

The transformation $h$ uses a trilinear interpolation to calculate the positions $x_{ijk}$. As the relative positions in a cell at which $f(x)$ is evaluated are the same for each cell, the interpolation fractions can be precalculated, both for $h$ and $J$.

In our case, selections are sets of grid nodes. To integrate over a selected area, we could switch to a cell-based interpretation of a selection, as we did with some selective visualization algorithms, see Section 4.2. However, we want to integrate over the selected set of nodes. Therefore, instead of adding all contributions of sample points in a cell to the integral of that cell, we add sample positions to the closest grid node. This is shown in Figure 5.4. Integrals using the Trapezium rule do not need interpolation of the data, only the nodes value is used. Linear
functions in $x, y$ and $z$ can be integrated exactly with the Trapezium rule. To integrate higher order functions exactly, one needs higher order quadratures. Using higher order quadratures takes more time, because the number of sample point increases rapidly. In practice, Trapezium integration suffices for areas consisting of many nodes. Integration of higher order functions (second moments, variance and covariance) for small areas (only a few nodes) requires higher order quadratures.

The integral routines have been implemented as an AVS-module \textit{(Volume Integral, see Appendix A)}. This module implements several sets of integrals over a selected region, and it generates one attribute set for each selected region. Among the implemented integrals are those listed on page 97. New integral sets can be added by editing the integral definition file and recompiling the module. The module offers the facility to switch between a trapezium rule, two-point Gauss-Legendre and four-point Gauss-Legendre quadrature rules.

5.2.2 Other techniques for attribute calculation

Until now, attribute sets have been viewed as sets of values that characterize a selection. Volume integrals fit nicely into this concept, as those are generic techniques to calculate characteristic values for selections. Attribute sets can, however, be considered to apply not only to selections, but also to other features in the data. In those cases, other techniques might be used to calculate attribute sets.

A first technique is interpolation. Interpolated data values for a position in a dataset can be considered as attributes for that position. This can e.g. be applied in combination with a critical point searching algorithm. The gradients at these critical points can be interpolated from a gradient field, e.g. resulting in a set of attributes. Classification of the critical points can then be done on the basis of the
attribute values, and they can be used for determining the right iconic mapping for a critical point.

Another approach is the use of positions along a curve, such as a streamline, as attribute sets. One can adjust a streamline technique to generate a streamline not only as a geometric object, but also as a series of attribute sets: one attribute set for each sampled position along the streamline. Such attribute sets contain information on a vector field, and can be used to visualize streamlines with iconic mappings.

Of course, many other techniques for attribute calculation can be developed and implemented. Virtually any visualization algorithm that preprocesses the data to search for specific information can be used to generate attribute sets. We restrict ourselves mainly to attribute calculation for selections, the techniques above were implemented to demonstrate that other techniques are compatible with this attribute calculation approach, and that the concept of attribute sets is applicable to more than just selective visualization.

5.2.3 Attribute set transformation

Attribute sets can be used for binding with the degrees of freedom of icons. Before the attributes are used in the iconic mapping stage, they can be transformed, to enrich their semantics, to reduce their size or to simplify their use in iconic mapping.

The main transformation we use for selections is ellipsoid fitting. The second moments of a selected region, and the variance/covariance matrix of a vector field in a selected region, define a tensor that can be associated with an oriented ellipsoid [26, 76]. The eigenvectors of the tensor correspond to the direction of the main axes of the ellipsoids, while the eigenvalues correspond to their length. To visualize such a tensor with an ellipsoid icon, the eigenvalues and eigenvectors of the tensor are calculated (using the Cyclic Jacobi algorithm, an iterative diagonalization process [65]).

A special transformation has been implemented for a streamline that is represented by a series of attribute sets. Each subsequent pair of attribute sets is transformed to a new attribute set that contains the line between the two positions on the streamline, and the derivatives of the data along the streamline at those positions. These attribute sets can be used by the iconic mapper to generate several different representations of the streamline.

These techniques are also implemented as AVS modules, see Appendix A for a description of the modules. In the appendix, we also list modules for attribute manipulation and administration, such as splitting and gluing of attribute sets, and reading, writing and viewing of attribute sets.
5.3 Iconic visualization mapping

Iconic visualization mapping is the stage in which an attribute set is mapped onto an icon. Below we discuss the use of icons in visualization, and the way we want to use icons. Then we describe a simple modelling language that we developed to construct icons, with their degrees of freedom bound to attributes, followed by some examples of icons that were developed with this language. The corresponding AVS module is Make Icon.

5.3.1 Icons

An icon as we use it, is a geometric object with parametric geometry and visual attributes that can be arbitrarily linked to data quantities. The function of an icon is to act as a symbolic representation, which shows essential characteristics or features of a data domain to which the icon refers. Reduction to essentials is the main purpose of the use of icons: to replace the original data by a symbolic representation that is more clear, compact, and meaningful, and which is related to the physical concepts and visual languages of an application [63].

Icons have been studied extensively in many fields, such as in theology and art history, in logic and the theory of signs or semiotics [18], and in pictorial information systems [9]. An attempt to relate the meaning of icons for visualization to classical sign theory has been made by Hesselink and Delmarcelle [29]. In scientific visualization, the icon concept is related to the Abstract Visualization Objects of Haber and MacNabb [26], and Parametric Graphic Objects used by van Wijk and van Liere [89] for computational steering. The term ‘glyph’ (literally: scratch) has also been frequently used [66], which is roughly comparable to the term icon as it is used here.

In graphic design, which has inspired scientific visualization, the use of symbolic representations is quite common. Early applications of iconic visualization can be found in statistical data visualization, such as Kleiner-Hartigan trees [8] and Chernoff faces [10]. These techniques are used for high-dimensional data that are defined in an abstract data space, and not in Euclidean physical space. Here, we concentrate on the use of icons for visualization of selections.

Of course, there is more to iconic visualization then the model we sketched and the tools we developed. Indeed, the similarity relation between the data visualized and the icon deserves attention, as this relation is also important for successful application of iconic visualization. A study of this subject would require thorough knowledge of the meaning of symbols. This knowledge should be combined with the visualization experiences of scientists, and their visualization semantics. This is beyond the scope of our study. In our examples, however, we used this similarity relation as a guideline in the development of the icons.
CHAPTER 5. ICONIC VISUALIZATION

5.3.2 Icon modelling language

We developed an icon modelling language to create icons from attribute sets. With this language, the icons geometry, colour, position and orientation can be made dependent on attribute values.

This modelling language is implemented as a set of C-macros. This has the advantage that icon descriptions can be compiled into the software, and thus icons can be generated quickly. A disadvantage is that this approach is not very user-friendly, and that the software has to be recompiled when a new icon is added. However, as our primary goal is to develop new visualization techniques, and to show how these techniques can be used, we did not want to develop a sophisticated icon design system, based on a geometric modelling approach. The C-macro approach proved to be well suited for our purpose, and we ended up with a large set of precompiled icons.

To provide complex icon shapes, we chose to describe icon geometries with parametric functions. These functions may depend on values of the attribute set for the icon. The definition of an icon is done in three steps:

1. a 2D contour definition (in the xy-plane). The contour may consist of parametric functions. Additionally, a colour, several colours or a colour map may be assigned to the contour.

2. a transformation of the 2D contour to a 3D geometry, by a rotation of the contour around the x, y or z-axis, or by a sweep along the z-axis, or by a sweep along an arbitrary trajectory.

3. a 3D transformation, consisting of scaling, translation and rotation.

Attribute values can be used in each of these steps by binding them to objects (functions) that can be used as arguments of the macros that specify the shape of the icon. In Appendix C, we list the macros that can be used for icon design.

5.3.3 Example icons

The icon modelling language has been used to create several icons. Some of these icons are described, to show the functionality of the language for some practical cases. The C-macro code for these icons is listed in Appendix C.

Arrows can be used to visualize a vector quantity at a specific position. The minimal set of degrees of freedom for a 3D arrow icon therefore is:

\[
\begin{align*}
    d_0 & \quad \text{x-position} & d_3 & \quad \text{x-direction} \\
    d_1 & \quad \text{y-position} & d_4 & \quad \text{y-direction} \\
    d_2 & \quad \text{z-position} & d_5 & \quad \text{z-direction}
\end{align*}
\]

The arrow is constructed by rotating the contour from Figure 5.5 around the x-axis. Then it is scaled in the x-direction with \( |(d_3, d_4, d_5)| \), to account for the
vector magnitude. Then, it is rotated to get the correct direction. The rotation angles $\alpha$, $\beta$ and $\gamma$ are determined from the direction vector. After rotation, the arrow is translated to $(d_0, d_1, d_2)$. The resulting arrow is also shown in Figure 5.5.

Ellipsoids can be used as a first order approximation of the shape of a selection, or as a first order approximation of a 3D distribution. An ellipsoid is determined by the position of its centre, the lengths of the main axes and the orientation of the main axes. The degrees of freedom are:

- $d_0$: $x$-position
- $d_1$: $y$-position
- $d_2$: $z$-position
- $d_3$: length axis$_0$
- $d_4$: length axis$_1$
- $d_5$: length axis$_2$
- $d_6$: angle $\alpha$
- $d_7$: angle $\beta$
- $d_8$: angle $\gamma$

To create an ellipsoid, we first create a sphere by rotating the contour from Figure 5.6 around the $x$-axis. Then this sphere is scaled with the lengths of its main axes, is rotated around the coordinate axis with angles $\alpha$, $\beta$ and $\gamma$, and is translated to its position $(x, y, z)$. The resulting ellipsoid is also shown in Figure 5.6. Ellipsoid icons can be created in many variants. Some example contours and corresponding icons for ellipsoids are shown in Figure 5.7 and 5.8.

A tube-like icon can be generated with a general sweep along a curve [6]. If the attribute set represents two positions, and the derivatives at those positions are known, a cubic Hermite interpolation curve can be used to generate the trajectory.
Figure 5.7: 2D contours for variants of an ellipsoid icon, see also resulting icons in Figure 5.8.

Figure 5.8: Icons for variants of an ellipsoid icon, see contours in Figure 5.7.
CHAPTER 5. ICONIC VISUALIZATION

for sweeping. An example of this icon is shown in Figure 5.9. Such icons are useful for visualization of e.g. streamlines, because of their 3D appearance, and opportunities to map additional data to the streamline (similar to techniques described by Schroeder [71]).

5.4 Examples of iconic visualization

We will give a simple example to demonstrate the iconic visualization process. Assume we want to visualize the region in a flow dataset where the velocity magnitude is large. Then, the first step is to select the regions with a large velocity magnitude, e.g. using the selection module with the following expression:

$$\text{select_out b= len(velo) > 0.5*max(len(velo))}$$

This region can be visualized with the techniques from Chapter 4, by constructing an isosurface around the region (Figure 5.10a), or by showing cross-markers at the selected nodes (Figure 5.10b). With the techniques described in this chapter, there is an alternative visualization approach: calculate the second moments of the region and map these values onto the degrees of freedom of an ellipsoid icon: Figure 5.10c.

If we also want to visualize the velocity in this region, we can use a hedgehog technique (a selective visualization: Figure 5.11a). The iconic approach is shown in Figure 5.11b: the average velocity is depicted by an arrow-icon, and the velocity distribution is depicted by an ellipsoid.

The differences between the two approaches are clear: the selective approach displays a large number of objects, while the iconic approach only displays a few objects. This reduction of objects results in a less accurate visualization (the cross-markers approximate the selected region more accurately than the ellipsoid). However, the reduction of objects also results in a less cluttered visualization. The arrow and ellipsoid visualization of the velocity is probably better than the hedgehog approach.

Another example is given in Figure 5.12. It is a ray traced image of geometries that were created with the AVS modules we developed. It shows the boundaries of a flow domain (of the blunt fin data set), some streamlines, and one streamline with flow probes along one of these streamlines. The boundaries are generated with a standard AVS module, the streamlines are generated with the selective streamline module, and the probes are an implementation of a probe for vector and vector gradients visualization [45] in the icon modelling language. The implementation of this complex probe icon appeared to be relatively easy with our icon modelling language.
Figure 5.9: Hermite interpolated tube icon.

Figure 5.10: Three visualizations of a selected region: a) a surface around the region; b) cross-markers at the selected nodes; c) an ellipsoid constructed from the second moments of the region
Figure 5.11: Two visualizations of velocity in a selected region: arrows at the selected nodes (upper image), and an arrow and ellipsoid visualizing the average velocity and the variance/covariance matrix of the velocity over the selected region (lower image).
CHAPTER 5. ICONIC VISUALIZATION

Figure 5.12: Visualization of blunt fin data; streamlines in combination with probes.

5.5 Conclusions

In this chapter, we have shown how selections can be visualized with iconic techniques. Iconic visualization was described as an extension to the selective visualization pipeline. We described how attribute values can be generated from selections with volumetric integrals, and how attribute sets can be prepared for iconic mapping. With these techniques, data can be visualized at a higher, more abstract level. We also described a modelling language with which icons can be developed, and with which a binding between the attributes and the degrees of freedom of the icon can be established. With the icon modelling language, a large set of icons was developed, and even complex icons such as the flow probe were implemented with little effort. The examples showed that iconic techniques allow for simple, comprehensive visualization of data.

This visualization approach is based on the assumption that not all data, or all selected data, should be visualized directly, but that attributes of selections can be sufficient for specific research purposes. For these cases, iconic visualization is a simple and effective visualization approach.

Iconic visualization is another step in the direction of more abstract visualization. The exploitation of iconic visualization requires an understanding of the relation between an icon and its meaning, i.e. how does one perceive an icon, and
how does one relate the icon to the data. For simple icons, such as an average velocity arrow, this relation is clear. For the development complex icons, this requires additional study and experimentation, both of the field of perception, as of the specific application areas of visualization.
Chapter 6

Case studies

The value of the techniques presented in Chapters 4 and 5 can only be shown by applying them to real datasets. In this chapter we present the results of experiments with four different cases: a flow in a backward facing step geometry, a turbulent flow in a pipe, an atmospheric flow with a chemical reaction and an atmospheric plume flow. For each of these cases, we will describe the background, the dataset, the question to be answered by visualization and the way in which visualization techniques were used to explore the data in order to answer the questions. The AVS modules that we developed and used for these cases studies are described in Appendix A.

6.1 Backward facing step

The first test case is a flow in a backward facing step geometry, that has been simulated by the Applied Mathematics Group of the Mathematics Department of Delft University of Technology. This simulation of a steady laminar flow is one of the 3D test cases of ISNaS, an Information System for Navier Stokes that is being developed at Delft University of Technology, in cooperation with several other research institutes in the Netherlands. The dataset is specified on a curvilinear grid of $25 \times 37 \times 9$ nodes, which is depicted in Figure 3.13a. At each node, pressure and velocity are given. The total dataset therefore consists of 8325 nodes, with seven numbers per node (three coordinates, a pressure and three velocity components).

The boundary conditions of the flow are simple: the velocity at the longitudinal boundaries is zero, the inlet is at the boundary near the step, and the outlet is the other end. In flows like these, a region of spiralling flow should occur just after the step. The goal of this visualization was to visualize streamlines that start at the inlet, then flow through the spiralling region, and then leave that region again.

This can of course be realized by manually probing the dataset at positions
where the flow enters the computational domain, but this would be tedious and difficult, as there are only few locations near the inflow boundary from where streamlines show spiralling behaviour.

The selective approach to this visualization consists of the following steps:

- find suitable starting positions for streamlines, using a selection expression
- trace streamlines from these positions
- visualize these streamlines

The streamlines have to cross the spiralling flow region, therefore this region is the target for selective visualization. A quantity that can be used to search for spiralling flow is normalized helicity density \( h_n \) [7]:

\[
h_n = \frac{\mathbf{v} \cdot (\nabla \times \mathbf{v})}{|\mathbf{v}| |\nabla \times \mathbf{v}|}
\]  

(6.1)

with \( \mathbf{v} \) velocity, and \( \nabla \times \mathbf{v} \) the curl of \( \mathbf{v} \).

The following expressions are used in the selection module cbs to select the region with high normalized helicity density:

![Diagram of AVS network for visualization of streamlines through region with high helicity density.](image)

**Figure 6.1:** AVS network for visualization of streamlines through region with high helicity density. read field reads the data, volume bounds creates the geometry for the grid boundaries, cbs-all creates the selected area, ndip_pick creates cross-markers at the selected nodes, cb_stream_sw traces streamlines through the selected region and geometry viewer displays the grid boundaries, the cross-markers at the selected nodes and the streamlines in one image.
CHAPTER 6. CASE STUDIES

Figure 6.2: Selective visualization of streamlines through region with high normalized helicity density

Figure 6.3: Iconic visualization of two streamlines and the regions with high normalized helicity density. (See also upper image on back cover.)
CHAPTER 6. CASE STUDIES

; calculate the curl (rotation)
bfs_rot a = rot(bfs_velo)

; calculate helicity density
bfs_hd a = dot(bfs_rot, bfs_velo)

; normalize helicity density
bfs_hdn a = bfs_hd / (len(bfs_velo)*len(bfs_rot))

; create the selection
select_out b = fabs(bfs_hdn) > 0.66

This selection is fed into the selective streamline tracer (cb_stream_sw); see the AVS-network in Figure 6.1. This module generates samples in the selected region, and traces streamlines from these samples (forward and backward). Extra modules are used to visualize the grid geometry (volume bounds) and the selection with cross-markers (ndip.pick). The resulting image is shown in Figure 6.2.

An alternative visualization is shown in Figure 6.3. There we use iconic visualization techniques to generate an image similar to Figure 6.2, but with fewer objects and with more information. The selection is now visualized by ellipsoids fitted around the two regions with high normalized helicity density, and the streamlines are visualized by coloured tubes. The radius of the tube is inversely proportional to the square root of the local velocity magnitude, and the colour corresponds to the pressure.

This case shows that selective visualization increases the visualization productivity. Manually probing the dataset would have taken several hours, while creating this visualization can be done in a few minutes. It also shows how this increase in visualization efficiency was achieved. A clear statement of the problem and some basic knowledge of the application field, such as knowing that regions with spiralling flow correspond to regions with a high normalized helicity density, could be transformed to input of the visualization process. This extra knowledge is used in the visualization process to speed up this process and to enhance the final visualization results. The iconic visualization uses the same techniques, and shows more data with fewer objects.

6.2 Turbulent pipe flow

The addition of small amounts of polymers to a turbulent flow may cause a large reduction in drag. This phenomenon is studied in the Laboratory for Aerodynamics and Hydrodynamics of Delft University of Technology [79]. Criteria are being developed to specify regions in which the polymer additives are assumed to be stretched by the flow, and become active in influencing the flow, leading to a global drag reduction.

Visualization techniques were used to compare such a criterion, $R$, with two
CHAPTER 6. CASE STUDIES

of its components, \( R_2 \) and \( \Pi_D \). \( \Pi_D \) contains information on the magnitude of
strain, \( R_2 \) contains information on the type of flow and \( R \), called the "elongation
parameter", contains mixed information on both magnitude of strain and type of
flow. \( R, R_2 \) and \( \Pi_D \) are related to the velocity \( \mathbf{v} \) as follows:

\[
L = \nabla \mathbf{v} \\
D = \frac{L + L^T}{2} \\
\Pi_D = \frac{\text{tr} (D \cdot D)}{2} \\
\Pi_D = \text{det} (D) \\
R = \frac{3\Pi_D}{\Pi_D} \\
R_2 = \frac{3\Pi_D}{\Pi_D^\frac{3}{2}}
\]

(6.2) \hspace{1cm} (6.3) \hspace{1cm} (6.4) \hspace{1cm} (6.5) \hspace{1cm} (6.6) \hspace{1cm} (6.7)

with \( \text{tr} (M) \) the trace, and \( \text{det} (M) \) the determinant of a matrix \( M \). The goal
of the visualization was to check whether \( R \) can be used as a criterion for the
relevant regions, or whether both \( \Pi_D \) and \( R_2 \) should be used separately.

The dataset for this visualization was a direct numerical simulation, consisting
of several time steps of the turbulent flow in a pipe. The grid is cylindrical, and
consists of more than thirty million nodes. The velocity data is represented on
a staggered grid. As the software expects the data to be represented at the grid
nodes, the data was interpolated to the cell centres in a preprocessing step, and
a new grid was defined that contains these cell centres (see Figure 6.4).

To calculate \( R, \Pi_D \) and \( R_2 \) we must calculate the partial derivatives matrix
\( L \). It appeared that calculating the derivatives from the interpolated data gave
severe accuracy problems. As the data in the dataset are not smooth, interpolation
followed by derivative calculation amounts to low-pass filtering the data.

Figure 6.4: Transformation of (2D) staggered grid to a grid with data defined
at the nodes.
before derivative calculation. Therefore the interpolation filter was extended with techniques that calculate the derivatives at the grid nodes directly from the original data. The filter also reads only a specific block from the data at one time step, as the entire dataset was much too large to be processed and visualized on the workstations. After this preprocessing, the input for the selective visualization techniques consists of a dataset for approximately fourteen thousand nodes, with velocity \texttt{velo} and the derivatives matrix \texttt{Lmat} specified at each node.

To investigate the relation between \( II_D \) and \( R_2 \), a region with elongational type of flow (\(|R_2| > 1\) was selected with the following expressions:

\[
\text{D a= .5*(Lmat + trans(Lmat))}
\]

\[
\text{IID a= .5*trace(mm(D,D))}
\]

\[
\text{IIID a= det(D)}
\]

\[
\text{R a= 3*IIID/IID}
\]

\[
\text{R2 a= R/sqrt(IID)}
\]

\[
\text{select_out b= fabs(R2) > 1}
\]

We visualized \( II_D \) in these regions by colouring the selected nodes with the value of \( II_D \), using a rainbow-like colormap from blue to red. To compare the combination of these criteria with \( R \), we added an isosurface at a high value for \( R \) (\(|R| = 80\)). The resulting images are shown in Figure 6.5. The regions with high magnitude of strain and elongational type of flow are the regions where the selected nodes are red. From the images it is clear that regions with high value for \( R \) include these regions. To validate this result, we also selected those regions where \( R \) is small, and both \( R_2 \) and \( II_D \) are large (those regions where \(|R|\) is less than 80 and \(|R_2|\) is greater than 1 and \( II_D \) is greater than 7500). This selection was empty, there were no nodes that satisfied the condition. From the visualization, it was concluded that \( R \) is a reasonable criterion to identify regions with large stretching, and may be used in computations instead of \( R_2 \) and \( II_D \) separately. This considerably simplifies the fluid flow computations.

Furthermore, we compared several other quantities with the region of high values of \(|R|\). This was done by visualizing these quantities at positions where their value was high, and comparing these visualizations to isosurfaces for high values of \(|R|\). We did this for the following quantities (with \( \omega = \nabla \times \mathbf{v} \), and \( D \)}
Figure 6.5: Combined visualizations of $H_D$, $R_2$ and $R$: upper image shows $H_D$ at regions where $|R_2| > 1$; lower image combines this with isosurfaces for $|R|$ at 80. It is clear that regions with a large value for $R$ correspond with the regions with a large value for $H_D$ and $R_2$. (See also Colour plate 1 on page 160.)
Figure 6.6: Visualizations for comparing several quantities with $|R|$: a) regions where $\omega \cdot D\omega$ is greater than 2e6; b) same, combined with an isosurface for $|R|$ at 80; c) regions where $\frac{\omega \cdot D\omega}{|\omega||D\omega|}$ is greater than 0.95; d) same, combined with an isosurface for $|R|$ at 80; e) regions where $\omega \cdot D\omega$ is greater than 27; f) same, combined with an isosurface for $|R|$ at 80. (See also Colour plate 2 on page 161.)
the symmetric part of $L$):

$$
\omega \cdot D \cdot \omega
$$

$$
\omega \cdot D \cdot \omega
$$

$$
|\omega| |D\omega|
$$

$$
\omega \cdot D \cdot \omega
$$

$$
\omega \cdot \omega
$$

All these quantities are related in some way to the so-called "enstrophy production", which is an important process in the dynamics of turbulent flow. For $\omega \cdot D\omega$, the following selection expressions were used:

```plaintext
; calculate omega
w a = rot(v)

; calculate w.Dw
wDw a = dot(w, mvm(D, W))

; select a region with high values for w.Dw
select_out b = wDw > 2e6
```

with 2e6 a suitable threshold value. For the other quantities, similar expressions were used. Resulting images are shown in Figure 6.6. It appears that $\omega \cdot D\omega$ shows only little correspondence to $R$, for $\frac{\omega \cdot D\omega}{|\omega||D\omega|}$ there is some resemblance to $R$, and $\omega \cdot D\omega$ corresponds quite well.

In this case study, we compared two scalar variables, $R_2$ and $II_D$, with a third scalar variable, $R$. By using these variables in different ways ($R_2$ as selection criterion, $II_D$ for volume rendering in the selection and $R$ for isosurface visualization), we were able to adequately use visualization to answer the question whether $R$ could be used instead of a combination of $R_2$ and $II_D$. Also, in comparing other quantities with the value for $R$, selective visualization proved to be useful. It appears that especially the visualization of relations between several quantities, and testing of hypotheses with respect to relations between several quantities, is straightforward using selective visualization techniques.

### 6.3 Atmospheric flow with a chemical reaction

The third test case concerns a dataset of an atmospheric flow with chemical reactions, computed at the Laboratory for Aero- and Hydrodynamics of Delft University of Technology. The region of the flow is a square area of $6.4 \times 6.4$ km$^2$ and 2.4 km high, in which NO diffuses from the earth's surface. This NO reacts with O$_3$ according to the following formula:

$$
NO + O_3 \rightarrow NO_2 + O_2
$$
Figure 6.7: AVS network for visualization of velocity at positions where NO₂ is formed. The middle branch (cbs-all and ndip_pick creates and visualizes the selection, the right branch (extract vector, field math, select_sampler and hedgehog) is for visualization of velocity with arrows at the selected region. The colour legend is created by the left branch (color legend and label).

The dataset contains one time step of a large-eddy simulation on a Cartesian grid of 42³ nodes. The dataset contains the two-way reaction concentrations of NO (N0), NO₂ (NO2) and O₃ (O3), the velocity components (velox, veloy, veloz) and the temperature (temp). For this dataset, we wanted to know the velocity at the regions where NO₂ is formed. It was expected that NO₂ would be formed in the region with upward flow only.

The amount of NO₂ formed is proportional to the reaction rate of the forward reaction. As the reaction rate is proportional to the product of NO and O₃ concentrations, we calculated this product:

\[
speed \text{ a} = \text{N0} \cdot \text{O3}
\]

and selected all nodes where a large amount of NO₂ is formed:

\[
\text{select}_\text{out b} = \text{speed} > .5*\text{max(speed)}
\]

The velocity at these positions was visualized with arrows (see the AVS network in Figure 6.7). The lower slices of the dataset were excluded from the visualization, as the region of interest was the eddying region, excluding the bottom slices. This was realized by adding " && k > 1 " to the selection expression:

\[
\text{select}_\text{out b} = \text{speed} > .5*\text{max(speed)} && k > 1
\]
Figure 6.8: Visualizations of velocity where reaction speed is high; upper image: arrows in coloured region; lower image: close-up. (See also Colour plate 3 on page 164.)
The resulting visualization is shown in Figure 6.8. The upper image gives an overview of the dataset, the lower image is a close-up.

A different visualization is shown in Figure 6.9: an iconic visualization of the same dataset. Instead of visualizing the velocity at each selected node, we clustered the selected nodes, calculated the average velocity and the velocity distribution over each cluster, and visualized these values with a large arrow (average velocity) and an ellipsoid icon (velocity distribution). The clusters are visualized with cross-markers, and each cross is coloured with the value of speed. From this image it is clear that the average velocity over the selected region is upward. As the distribution ellipsoid is relatively small compared to the average velocity arrows, it can be concluded that velocity in the selected regions is generally upward.

Just as in the previous case, it was simple to answer the questions with respect to the data with selective visualization techniques. The hypothesis was easily transformed to a selection expression, and the visualization answered the questions. The relation between several quantities could be visualized effectively, the visualization clearly showed that velocity is upward where NO₂ is formed. With the iconic techniques, we were able to create a visualization that answers the question, but with less objects. This symbolic visualization probably shows even more clearly that velocity is upward at regions where NO₂ is formed.

### 6.4 Atmospheric flow with plume

The last case study concerns another large-eddy simulation from the Laboratory for Aero- and Hydrodynamics of Delft University of Technology. The development of a plume is simulated in a rectangular region of $4 \times 2 \times 1.5$ km$^3$. The origin of the plume is at one of the boundaries of the box (near the lower left corner of the back face in Figure 6.10). From this position, the material released is spread by convection and diffusion. Simple prediction models for concentration are based on the following relation between the turbulent mass flux and the concentration gradient for estimating the distribution of material:

$$u'_i c' = -K \frac{\partial c}{\partial x_i}$$  \hspace{1cm} (6.8)

with $u'_i c'$ component $i$ of the turbulent mass flux, being the product of the velocity fluctuation $u'_i$ and the concentration fluctuation $c'$, $c$ the concentration and $K$ a constant. Both the turbulent mass flux and the concentration gradient can be derived from the quantities that are calculated in the simulation. The goal of the visualization was to test whether the relation described by Equation 6.8 holds for this case.

The dataset we used was calculated on a staggered grid of $100 \times 100 \times 100$ nodes and contains the velocity and concentration (velocity components are given at the cell boundaries, concentration values are given at the cell centres) for two time
Figure 6.9: Iconic visualization of regions where NO₂ is formed, average velocity over a region is visualized with large arrow, velocity distribution is visualized with an ellipsoid, regions are visualized with cross-markers. (See also middle image on back cover.)

Figure 6.10: Visualization of $K$ throughout the plume, using coloured cross-markers. (See also Colour plate 4 on page 164.)
CHAPTER 6. CASE STUDIES

steps. From this dataset, we calculated the turbulent mass flux at the boundaries of cell \([i, j, k]\) with

\[
    u'c'[i, j, k] = \frac{1}{2} (u_2[i, j, k] - u_1[i, j, k]) \\
    \{ (c_2[i, j, k] + c_2[i-1, j, k]) - (c_1[i, j, k] + c_1[i-1, j, k]) \} \quad (6.9)
\]

with \(u'c'\) the \(x\)-component of the turbulent mass flux, \(u_2\) the \(x\)-component of the velocity on the second time step, \(c_2\) the concentration on the second time step and \(u_1\) and \(c_1\) corresponding values on the first time step. For \(v'c'\) and \(w'c'\), similar formulas were used.

The concentration gradient (for the first time step) is calculated at the cell boundaries by differencing:

\[
    \frac{\partial c}{\partial x}[i, j, k] = \frac{(c_1[i, j, k] - c_1[i-1, j, k]) - (c_2[i, j, k] - c_2[i-1, j, k])}{2\Delta x} \quad (6.10)
\]

with \(\Delta x = 40\) m. Similar formulas were used for \(\frac{\partial c}{\partial y}\) and \(\frac{\partial c}{\partial z}\). The calculation of the turbulent mass flux (tmf) and the concentration gradient (cg) were done in a preprocessing step.

For visualization, we used two approaches to visualize the relation between tmf and cg:

![Visualization of dotprod throughout the plume, using coloured cross-markers. (See also lower image on back cover.)](image)

Figure 6.11: Visualization of dotprod throughout the plume, using coloured cross-markers. (See also lower image on back cover.)

124
CHAPTER 6. CASE STUDIES

- compare the magnitudes of tmf and cg. This can be done by calculating the value for K (Equation 6.8) with expression:

\[ K = -1 \times \text{len(tmf)} / \text{len(cg)} \]

and visualizing this field for the plume. We selected the plume with

\[ \text{select_out b = conc > .0001} \]

as the concentration is very small outside the plume. A resulting visualization is shown in Figure 6.10. K is coloured, the colour varies for values of K from -1 to 0, the value of K ranges from -10.2 to 0, and the average value for K is -0.022.

- compare the directions of tmf and cg. This can be done by calculating the normalized dot product of tmf and cg:

\[ \text{dotprod a = dot(tmf, cg) / (len(tmf) * len(cg))} \]

Again, this value can be visualized in the plume, see Figure 6.11. Alternatively, we can visualize this value at regions where the magnitude of either tmf or cg is relatively large, thus removing those nodes where the dot product might be inaccurate because the vectors are relatively small, Figure 6.12.

For both visualizations, it is clear that the relation from Equation 6.8 does not hold for this simulation. The value of K varies significantly over the plume. Also, there is little correlation between the directions of tmf and cg, as dotprod varied from -1 to 1 over the plume. Even for large vectors of tmf and cg, we could not find a relation like the one in Equation 6.8 for tmf and cg.

It is, of course, beyond the scope of this case study to go into the details why this relation did not hold for this case. Our interest lies in the fact that we were able to adequately create visualizations that give an answer to the questions that were posed. Again, in the case of visualization of the relation between two variables, and with a clear hypothesis, the selective techniques were very effective. With these techniques we were able to visualize the relation in straightforward ways, and we did not need any probing techniques.

A final view of this dataset is shown in Figure 6.13. With the H-buffer algorithm [83], we created a different view on the dataset, by visualizing the concentration field as a density field, combined with two semi-transparent isosurfaces for the concentration field. This view can be used to test whether the simulation of a plume indeed looks like a plume, and in that way contributes to verification of the simulation data.
Figure 6.12: Visualization of dotprod throughout the plume at nodes where the magnitude of tnf or cg is relatively large. (See also Colour plate 5 on page 165.)

Figure 6.13: Another view on the plume dataset: volume rendering of a concentration field, combined with semi-transparent isosurfaces.
6.5 Conclusions

We applied selective and iconic visualization techniques for four different CFD datasets. For each case, a clear research question was formulated, which was easily transformed to a set of selection expressions. With these selection techniques, we were able to create visualizations that were useful in answering the research questions.

As for the timing and interaction aspects of the techniques, all functions could be performed within a few seconds (on SGI Onyx), except the gradient calculations on curvilinear grids. These timings include parsing of a selection expression and evaluation of the expression for a dataset. On the same workstation, the resulting geometries could be viewed and rotated interactively within the AVS Geometry Viewer. These fast processing times allow for interactive exploration of threshold selection and selection specification.
Chapter 7

Conclusions

Three subjects with respect to selective visualization on curvilinear grids are discussed in this thesis:

- Chapter 3 deals with techniques for curvilinear grids,
- Chapter 4 deals with techniques for selective visualization and
- Chapter 5 deals with techniques for iconic visualization

Below, we draw conclusions from the material presented in these chapters, and give directions for future research.

7.1 Techniques for curvilinear grids

There is no simple, global relationship between a physical space position in a 3D curvilinear grid and the cell indices and fractional offsets of that position. This is a problem for visualization of curvilinear grids, as for many visualization techniques the cell indices and the fractional offsets for physical space positions are required. Techniques that can be used to calculate the cell number and fractional offsets are the subject of Chapter 3.

One approach to solve these point location problems is to use a computational space, a uniform Cartesian grid to which the curvilinear grid and data are transformed. This, however, introduces other problems with respect to the continuity of the data. Therefore, our goal was to develop new techniques and compare them with existing techniques that operate in physical space.

For point location, we describe three alternatives: an iterative technique, Stencil Walk, and two geometric techniques: one that uses bilinear patches as cell boundaries, and one that decomposes cells into five tetrabedra. It appears that Stencil Walk is the best (fastest and robust) technique for initial point location,
while tetrahedrization is the most efficient technique for incremental point location, provided that more than one step per cell is taken.

For interpolation, trilinear interpolation is assumed to be the standard interpolation if the data varies linearly in cells. Trilinear interpolation uses fractional offsets of a position in a cell, which can be calculated by an iterative algorithm such as Stencil Walk. We describe two alternative interpolation techniques to be used in conjunction with the tetrahedral point location technique. One of these techniques, TetLin, linearly interpolates data in a tetrahedron, the other technique, TetTri, makes an approximation of the fractional offsets, based on the tetrahedrization. From experiments with two CFD datasets, it appeared that the average relative interpolation error of TetTri is an order of magnitude less than the average relative interpolation error of TetLin. When we compare these interpolation errors with the integration error of a second order Runge Kutta integration, it appears that the TetTri interpolation error becomes significantly larger than the integration error if many steps (more than approximately 2.5) per cell are taken. If less than one step per cell is taken, the TetLin interpolation error is less than the integration error, thus TetLin interpolation should be enough in those cases.

If we consider timing results of these techniques, we can conclude that Stencil Walk is faster if the number of steps per cell is less than .75 (second order Runge Kutta integration has two point location steps per integration step, thus for second order Runge Kutta integration this value is .4). So, for fast, relatively accurate streamline integration, tetrahedrization with TetTri interpolation should be used. More accurate streamlines can be obtained at much more computational costs, by using Stencil Walk with TRILIN interpolation.

7.2 Selective visualization

Visualization of all data of large 3D numerical simulations is nowadays not feasible anymore, for there is too much data to visualize. The selective visualization approach is a solution to this problem. It consists of two important stages: selection creation and selective mapping. In the selection creation stage, a (Boolean) expression is evaluated for all nodes of the dataset, yielding a set of selected nodes: a selection. This selection is used in the visualization mapping stage, either to be visualized directly, for clipping or for parameter extraction. In an intermediate stage, selections can be clustered or manipulated with simple image processing techniques.

The power of this approach lies in the flexible creation of selections and the various functions of selections in the visualization mapping. It offers many possibilities to selectively visualize only the data that are interesting. The advantages of selective visualization over standard visualization techniques is that more relevant data can be displayed without cluttering the display, i.e. more information can be deduced from the visualization. Especially in those cases, where there is a
clear question or hypothesis with respect to the data, this visualization approach is efficient and effective, as is shown in the case studies.

7.3 Iconic visualization

In the selective visualization approach, no special meaning is attached to the selected region: there is no conceptual difference between the selection of an index space layer, or a region with a high helicity density. The iconic visualization approach extends selective visualization with an abstraction step. Selections are no longer merely viewed as sets of selected nodes, but (clustered) selections are viewed as features that deserve a special treatment in the visualization process. To describe these features, attribute sets are introduced. The attribute sets play a role similar to the role of selections in selective visualization: they are generated in a preprocessing step and, after a transformation step, carry information that is used to enhance the visualization mapping.

For the generation of these attribute sets, generic techniques such as integration or interpolation can be used. After their generation, attribute sets can be transformed, e.g. scaled or mapped to another coordinate system, to prepare the attribute set for visualization mapping.

In the iconic mapping stage, the attributes of an attribute set are mapped on the the degrees of freedom of an icon. Icons are described with a modelling language, with which the icons geometry and other visual parameters can be expressed in terms of elements of an attribute set.

Application of iconic visualization techniques requires even more specific knowledge of the dataset than application of selective visualization, as the selection should represent some feature or phenomenon in the data. It also might require special techniques to select phenomena that cannot be found by our general selection techniques.

Because of the abstraction step in the iconic visualization process, visualization show less data than is shown with selective visualization, but they still may carry the same amount of information. However, also because of the abstraction step, knowledge will be required to comprehend an iconic visualization. Visualizations created with iconic visualization techniques will not be as obvious as many visualizations created with other techniques.

7.4 Future research

In Chapter 3 we discussed physical space techniques for grid traversal and interpolation for 3D curvilinear grids. Techniques that operate in computational space are also known, and in Section 3.2 we describe that transforming a vector field to computational space may result in a discontinuous vector field, which was the reason for us to work on physical space alternatives. We did not perform a
CHAPTER 7. CONCLUSIONS

comparison of these approaches. However, a quantitative comparison of both the efficiency and accuracy of physical space techniques versus computational space techniques should be applied to decide whether discontinuous vector field errors are worth the supposed gain in efficiency.

Many visualization techniques are not “error-free”. Streamline integration techniques, for example, will have integration errors (see Chapter 3), and isosurface techniques may position isosurfaces at wrong positions because of interpolation errors. Until now, little attention has been paid to the visualization of these errors. However, visualization of “uncertainties”, caused by the data or the visualization process, would make visualizations more faithful. For this, “fuzzy” visual primitives can be developed. For streamlines, techniques similar to the visualization of turbulent flow [30] or convection and diffusion [50] might be used for these purposes.

In Chapter 3 we also discussed that it might be advantageous to directly use a staggered grid in the visualization process if the data has been generated on a staggered grid. However, most visualization techniques assume a structured (or even Cartesian) grid, with data specified at the nodes. More efficient and more accurate visualization techniques might be developed if the original output of the simulation can be used, instead of filtering or converting it to a suitable format for visualization. Other grid types that deserve special techniques are unstructured grids, time-dependent grids and multi-block grids.

In Chapter 4 and 5 we described general techniques that were developed to diminish the amount of data to be displayed by visualization algorithms, yet maintaining, or even increasing, the amount of information in the final visualization. Our aim was to develop generic techniques, and therefore these techniques might not be well suited to extract some (application-specific) features, or physical phenomena. Several studies exist that describe work for visualization of specific phenomena such as turbulence [30], vortices in turbulent flow [81, 3] and flow topology [28, 23]. Such features cannot be easily visualized with our techniques. It would be worth while to try to bring both approaches: selective and iconic visualization, and specific search and visualization techniques, closer to each other. A means for this is extending the current selective and iconic visualization techniques with more powerful techniques and control structures, that allow for iterating over a selection and attribute calculation process, to converge automatically to a better visualization.
References


REFERENCES


REFERENCES


REFERENCES


136
REFERENCES


137
REFERENCES


Appendix A

AVS Modules

This appendix contains a short description of the AVS modules that were developed as part of the work described in this thesis. AVS modules are divided in four types: data input, filter, mapper, data output. Data input modules generate data, or read data from disk, filter modules filter data, mapper modules transform data to geometries (i.e. objects that can be displayed), and data output modules display data, or write it to disk. The types of the modules are shown in Table A.1.

<table>
<thead>
<tr>
<th>Input</th>
<th>Filter</th>
<th>Mapper</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make AV</td>
<td>cbs-all</td>
<td>cb_iso tetra</td>
<td>AV Viewer</td>
</tr>
<tr>
<td>Read AV</td>
<td>cb_sampler</td>
<td>cb_sample_axis</td>
<td>Write AV</td>
</tr>
<tr>
<td>Glue AV</td>
<td></td>
<td>cb_sample_dot</td>
<td></td>
</tr>
<tr>
<td>ndip.dilate_eros</td>
<td></td>
<td>cb_select_axis</td>
<td></td>
</tr>
<tr>
<td>ndip.open_close</td>
<td></td>
<td>cb_select_dot</td>
<td></td>
</tr>
<tr>
<td>ndip.pick</td>
<td></td>
<td>cb_slice</td>
<td></td>
</tr>
<tr>
<td>Process AV</td>
<td></td>
<td>cb_stream...</td>
<td></td>
</tr>
<tr>
<td>Sample AV</td>
<td></td>
<td>Make Icon</td>
<td></td>
</tr>
<tr>
<td>Sample Volume</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split AV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stream to Hermite</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volume Integral</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: AVS modules and their types

Short descriptions of these modules follow, in alphabetical order per module type.

A.1 Input modules

Make AV The Make AV module generates an attribute set. The maximum size
APPENDIX A. AVS MODULES

of the attribute set is 15 elements. If attribute sets with more elements are needed, two Make AV modules can be used, and the outputs can be concatenated together using Glue AV.

Read AV module reads attribute sets from a disk file.

A.2 Filter modules

cbs-all is a macro module, that contains a nice user interface to the module that generates selections: cbs. This module calculates derived fields, selections and scalar or vector variables by evaluating an expression for each node of the dataset. The input data field has to be on a regular or curvilinear grid. The derived fields, selections and variables are stored in the module, and can be put on one of the output ports. In addition, derived data can be dumped to disk, in either C or Fortran order, and either in binary or ASCII format. If one of the input ports changes (or an expression changes), or one of the parameters changes, the module performs the following actions:

- the expression (or expressions, if read from a file) are evaluated, and commands executed.
- if an expression corresponds to one of the output ports, it is stored.
- if one of the elements of the expressions for the output ports have changed, the expressions for the corresponding output port are reevaluated.

cb_sampler generates samples from a selection, and stores the selected positions in an irregular grid.

Glue AV glues attribute sets. The attribute sets connected to the input-ports are concatenated together to one attribute set. On each input port the same number of attribute sets must be provided.

ndip_dilate_erode dilates or erodes a selection. A dilation operation is a growing operation. A mask is applied to each node of the grid. If one of the nodes below the mask is selected, the node to which the mask is applied, is set selected. An erosion is the complementary operation: a mask is applied to each node of the grid, and if one of the nodes below the mask is not selected, the node to which the mask is applied is set to not selected.

ndip_open_close opens or closes a selection. An open operation is an erosion, followed by a dilation. A close operation is a dilation, followed by an erosion. (A dilation operation is a growing operation. A mask is applied to each node of the grid. If one of the nodes below the mask is selected, the node to which the mask is applied, is set selected. An erosion is the complementary operation: a mask is applied to each node of the grid, and if one of the
nodes below the mask is not selected, the node to which the mask is applied is set to not selected.)

ndip_pick clusters the input selection, based on the connectivity specified. Clusters that contain less than min_size nodes are not put on the output port. The dial select can be used to select on of the clusters, if select is 0, all cluster are put on the output port. (A cluster field is just a byte field, where each nodes value is the nodes cluster number, 0 means no cluster.) Alternatively, selecting a node in the geometry viewer window will select the cluster to which that node belongs. Selecting a node if only one cluster is output will unselect that cluster, i.e. all cluster are output again. The current cluster number is reflected in the select parameter value. The geometry output are crosses or dots at the cluster nodes, the cluster nodes can be coloured with the value of a scalar field (a colouromap should be provided in that case).

Process AV processes attribute sets. There are three modes of operation: Inherit, Transform and Extract. In the Inherit mode, the module performs a transformation that is described in the attribute-set structure, if possible. In the Transform mode a transformation can be chosen from a list of transformations. Each attribute set at the input port is transformed, and put at the output port. In the Extract mode, an selection of elements on the input attribute sets can be made. The size of the output attribute sets must be given. Then the numbers of the elements of the input attribute sets that must be included in the output attribute set are given. From each attribute set at the input port, this selection is made.

SampleAV selects one attribute set from the input port, and puts it on the output port.

Sample Volume calculates position and data from a series of points in computational space, by interpolating trilinearly. The series of points is represented as a string formatted as attribute sets.

Split AV splits all attribute sets at the input port into chunks of a user specified size. These chunks are put on the output ports.

Stream to Hermite takes samples from a sequence of attribute sets, and generates attribute sets that can be used for Hermite interpolation. First, the number of the attribute sets at the input port is reduced by sampling down to the number given in the user interface. Then, for each two successive attribute sets of the remaining attribute sets an attribute set is generated. This attribute set consists of the first of the successive attribute sets, the second, the derivatives at the first attribute set and the derivatives at the second. The derivatives are approximated with central differences.
APPENDIX A. AVS MODULES

Volume Integral integrates pre-defined sets of functions, over selected and clustered parts in a dataset. This set of functions can be chosen from the user interface. The accuracy can be selected from the user interface; Trapezium quadrature, 2-points Gauss quadrature and a 4-points Gauss quadrature. For each cluster an attribute set is generated, and this attribute set has as elements the integrated functions over the cluster.

A.3 Mapper modules

cb_iso_tetra generates an isosurface that is clipped against a selection (if a selection is specified). The isosurface is constructed using a tetrahedrization of the grid.

cb_sample_axis generates an axis at each node of the samples field.

cb_sample_dot generates a dot at each node of the samples field.

cb_select_axis generates an axis at a selected node, by drawing six lines from the selected node, one to each of the neighbouring nodes. The length of the lines is determined by scale (if scale is 2., the line ends at the neighbouring node). If both a scalar field and a colon map are specified, the axes are coloured with the scalar value of the node. The grid of the selection and the scalar field have to be identical (i.e. the dimension should be equal).

cb_select_dot generates a dot (a zero length line) at a selected node. If both a scalar field and a colon map are specified, the dot is coloured with the scalar value of the node. The grid of the selection and the scalar field have to be identical (i.e. the dimension should be equal).

cb_slice generates orthogonal slices in index space (i.e. along grid lines) through selections. For the cluster specified by cluster_no (or all clusters, if cluster_no is zero), the cluster centre is calculated and slices are generated through the cluster centre. The slices are clipped against the cluster boundaries.

cb_stream... is a collection of streamline modules, that use different interpolation and/or cell traversal techniques, but that are otherwise the same. All modules calculate streamlines in, through or outside a selected region. The selected region is defined by a boolean dataset on a grid (value 0 means a node is not selected, other values mean that a node is selected), and by cell_crit, which specifies the minimum number of nodes of a cell that have to be selected for a cell to be selected. The parameter where determines where the streamlines have to be generated: in, through or outside a selected region. Starting positions for the streamline are generated at (or near) selected nodes if where is "in" or "through", and at (or near) not selected nodes if where is "outside". If where is "through", streamlines are traced
until the maximum trace time is reached, in "in" mode, tracing is stopped if a streamline enters a not selected cell, and in "out" mode, tracing is stopped if a streamline enters a selected cell (and of course, if the maximum trace time is reached). Streamlines can be traced forward and backward from the starting position (b.time and f.time). One of the streamlines can be output as an attribute vector (of computational space coordinates), and all starting positions are output as a samples field. The module uses a second order Runge-Kutta integration technique, interpolation is discussed at the bottom of this manual page.

Make Icon The Make Icon module generates geometry from attribute sets. For each attribute set on the input port an icon is generated. The icons are modelled with an icon modelling language. If ‘view icon’ is selected an icon can be tested, the parameters can be changed and an icon is generated instantly.

A.4 Output modules

AV Viewer display contents of attribute sets in a scrollable window.

Write AV writes attribute sets to a disk file.
Appendix B

Syntax of input for cbs module

This appendix contains the syntax for the input string of cbs, that is described in Chapter 4. The syntax is given in BNF, and is derived directly from the yacc input file. The tokens used in the syntax description are taken from the lex input file. Terms that are defined further are in the TERM typeface, while tokens are in the token typeface.

According to the following definition of the syntax of the input for the cbs-module:

\[ \text{INPUT ::= ASSIGNMENT} \mid \text{QUERY} \mid \text{COMMAND} \]

the input string may contain one of the following:

- an assignment expression to be evaluated, resulting in a new field or variable
- a query concerning the data, that results in a value presented to the user face
- a command to be executed by the module

Each of these types will be discussed in the following sections, followed by a section on the syntax definition of arithmetic expressions.

B.1 Expressions

For expressions, the following syntax applies:

\[ \text{ASSIGNMENT ::= BOOLEAN-ASSIGNMENT} \mid \text{ARITHMETIC-ASSIGNMENT} \mid \text{CONSTANT-ASSIGNMENT} \]

\[ \text{ARITHMETIC-ASSIGNMENT ::= IDENTIFIER a= ARITHMETIC} \]
APPENDIX B. SYNTAX OF INPUT FOR CBS MODULE

BOOLEAN-ASSIGNMENT ::= IDENTIFIER b= BOOL-EXPRESSION

CONSTANT-ASSIGNMENT ::= IDENTIFIER c= ARITHMETIC
| PARAMETER c= ARITHMETIC

BOOL-EXPRESSION ::= IDENTIFIER
| ( BOOL-EXPRESSION )
| ! BOOL-EXPRESSION
| BOOL-EXPRESSION && BOOL-EXPRESSION
| BOOL-EXPRESSION || BOOL-EXPRESSION
| COMPARISON

COMPARISON ::= ARITHMETIC < ARITHMETIC
| ARITHMETIC > ARITHMETIC
| ARITHMETIC <= ARITHMETIC
| ARITHMETIC >= ARITHMETIC
| ARITHMETIC == ARITHMETIC
| ARITHMETIC != ARITHMETIC

B.2 Commands

The syntax of the commands that can be executed by cbs is as follows (the effect of the commands is shown in Table B.1):

COMMAND ::= list
| logfile IDENTIFIER
| logfile
| loging INTEGER
| rem IDENTIFIER
| read IDENTIFIER
| dac IDENTIFIER
| daf IDENTIFIER
| dbc IDENTIFIER
| dbf IDENTIFIER
| time INTEGER
| wait
| continue

B.3 Queries

For queries, the following syntax rule applies:
<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>logfile file</td>
<td>Set logfile to file</td>
</tr>
<tr>
<td>logging 1/0</td>
<td>Turn logging on/off</td>
</tr>
<tr>
<td>rem field</td>
<td>Remove field</td>
</tr>
<tr>
<td>read file</td>
<td>Read expressions from file</td>
</tr>
<tr>
<td>dac field</td>
<td>Write field in C-order, ASCII format to file</td>
</tr>
<tr>
<td>daf field</td>
<td>Write field in Fortran-order, ASCII format to file</td>
</tr>
<tr>
<td>dbc field</td>
<td>Write field in C-order, binary format to file</td>
</tr>
<tr>
<td>dbf field</td>
<td>Write field in Fortran-order, binary format to file</td>
</tr>
<tr>
<td>time 1/0</td>
<td>Turn timing output on/off</td>
</tr>
<tr>
<td>wait</td>
<td>Suspend reading expressions from file</td>
</tr>
<tr>
<td>cont</td>
<td>Continue reading expressions from file</td>
</tr>
</tbody>
</table>

Table B.1: Available commands and their effect for cbs

QUERY ::= ARITHMETIC

Note that queries cannot be evaluated for a whole data field, queries should result in one scalar or vector constant.

B.4 Arithmetic

Arithmetic expressions are defined according to the following rules:

ARITHMETIC ::= IDENTIFIER | CONSTANT | COORDINATES | INDICES
              | PARAMETER | ( ARITHMETIC )
              | ARITHMETIC * ARITHMETIC
              | ARITHMETIC / ARITHMETIC
              | ARITHMETIC + ARITHMETIC
              | ARITHMETIC - ARITHMETIC
              | UNARY-FUNC | BINARY-FUNC
              | STAT-FUNC | GRAD-FUNC

UNARY-FUNC ::= cos(ARITHMETIC)
              | elog(ARITHMETIC)
              | exp(ARITHMETIC)
              | fabs(ARITHMETIC)
              | log(ARITHMETIC)
              | len(ARITHMETIC)
              | norm(ARITHMETIC)
              | sin(ARITHMETIC)
APPENDIX B. SYNTAX OF INPUT FOR CBS MODULE

| \text{sqrt}(\text{ARITHMETIC}) |
| \text{sqrt}(\text{ARITHMETIC}) |
| \text{trace}(\text{ARITHMETIC}) |
| \text{trans}(\text{ARITHMETIC}) |
| \text{det}(\text{ARITHMETIC}) |

\text{BINARY-FUNC ::= dot}(\text{ARITHMETIC}, \text{ARITHMETIC})
| \text{cross}(\text{ARITHMETIC}, \text{ARITHMETIC}) |
| \text{mm}(\text{ARITHMETIC}, \text{ARITHMETIC}) |
| \text{mvm}(\text{ARITHMETIC}, \text{ARITHMETIC}) |
| \text{vmm}(\text{ARITHMETIC}, \text{ARITHMETIC}) |

\text{STAT-FUNC ::= max}(\text{ARITHMETIC})
| \text{min}(\text{ARITHMETIC}) |
| \text{ave}(\text{ARITHMETIC}) |
| \text{rms}(\text{ARITHMETIC}) |
| \text{maxr}(\text{BOOL-EXPRESSION}, \text{ARITHMETIC}) |
| \text{minr}(\text{BOOL-EXPRESSION}, \text{ARITHMETIC}) |
| \text{aver}(\text{BOOL-EXPRESSION}, \text{ARITHMETIC}) |
| \text{rmsr}(\text{BOOL-EXPRESSION}, \text{ARITHMETIC}) |

\text{GRAD-FUNC ::= div}(\text{IDENTIFIER})
| \text{rot}(\text{IDENTIFIER}) |
| \text{grad}(\text{IDENTIFIER}) |
| \text{ii}(\text{IDENTIFIER}) |

\text{COORDINATES ::= node | xc | yc | zc}

\text{INDICES ::= index | i | j | k}

\text{PARAMETER ::= ipd1 | ipd2 | ipd3 | ipd4 | ipd5}
| fpd1 | fpd2 | fpd3 | fpd4 | fpd5 |

\text{SCALAR-CONSTANT ::= INTEGER | DOUBLE}

\text{VECTOR-CONSTANT ::= INTEGER-VECTOR-CONSTANT}
| DOUBLE-VECTOR-CONSTANT |

\text{INTEGER-VECTOR-CONSTANT ::= [INTEGER,INTEGER]}
| [INTEGER,INTEGER,INTEGER] |
| [INTEGER,INTEGER,INTEGER,INTEGER] |
| [INTEGER,INTEGER,INTEGER,INTEGER,INTEGER] |
APPENDIX B. SYNTAX OF INPUT FOR CBS MODULE

DOUBLE-VECTOR-CONSTANT ::= [DOUBLE,DOUBLE]
                         | [DOUBLE,DOUBLE,DOUBLE]
                         | [DOUBLE,DOUBLE,
                               DOUBLE,DOUBLE]
                         | [DOUBLE,DOUBLE,DOUBLE,
                               DOUBLE,DOUBLE,DOUBLE,
                               DOUBLE,DOUBLE,DOUBLE]

The meaning of the unary, binary, statistical and gradient functions is shown in Table B.2.

Some additional comments on the input:

- Maxima, minima, averages and root mean squares of vectors are calculated per component of the vector.

- Integers are just a series of digits (with or without a preceding sign), while doubles can be specified with a comma, or in scientific notation (.314159265e+1), also with or without a preceding sign.

- An IDENTIFIER is any string of characters that does not contain spaces or quotes. To prevent an IDENTIFIER (such as a filename that begins with a slash) to be interpreted as an expression, IDENTIFIERS that might match an expression should be enclosed in double quotes (") to prevent this misinterpretation. The same holds for identifiers that contain strange characters. For example, to use read a file named /tmp/logfile.log, the following expression should be used:

  read "/tmp/logfile.log"

and to calculate the divergence of a field named ‘div’, the following expression should be used:

  divdiv a= div("div")

In this last case, the quotes are necessary to prevent the field name to be interpreted as an operator.

- Spaces are generally not required in expressions, but they may enhance readability of expressions.

- The precedence of operators is the same as in the C programming language. When in doubt, use parenthesis to specify the order of evaluation of subexpressions.

- The Identifiers select_out, field_out, smin_out and smax_out should not be used in the right hand side of expressions. When used in the left hand side, the resulting data will be put on the corresponding output port of the module.
### APPENDIX B. SYNTAX OF INPUT FOR CBS MODULE

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>cos()</td>
<td>Cosine of a scalar</td>
</tr>
<tr>
<td>elog()</td>
<td>Logarithm of a scalar</td>
</tr>
<tr>
<td>exp()</td>
<td>Exponent of a scalar</td>
</tr>
<tr>
<td>fabs()</td>
<td>Absolute value of a scalar</td>
</tr>
<tr>
<td>log()</td>
<td>10-based logarithm of a scalar</td>
</tr>
<tr>
<td>len()</td>
<td>Length of a vector</td>
</tr>
<tr>
<td>norm()</td>
<td>Normalized vector</td>
</tr>
<tr>
<td>sin()</td>
<td>Sine of a scalar</td>
</tr>
<tr>
<td>sqr()</td>
<td>Square of a scalar</td>
</tr>
<tr>
<td>sqrt()</td>
<td>Square root of a scalar</td>
</tr>
<tr>
<td>trace()</td>
<td>Trace of a matrix</td>
</tr>
<tr>
<td>trans()</td>
<td>Transpose of a matrix</td>
</tr>
<tr>
<td>det()</td>
<td>Determinant of a matrix</td>
</tr>
<tr>
<td>dot(),</td>
<td>Dot product of two vectors</td>
</tr>
<tr>
<td>cross(),</td>
<td>Cross product of two vectors</td>
</tr>
<tr>
<td>mm(),</td>
<td>Multiplication of two matrices</td>
</tr>
<tr>
<td>mvm(),</td>
<td>Matrix vector multiplication</td>
</tr>
<tr>
<td>vmm(),</td>
<td>Vector matrix multiplication</td>
</tr>
<tr>
<td>max()</td>
<td>Maximum of a field</td>
</tr>
<tr>
<td>min()</td>
<td>Minimum of a field</td>
</tr>
<tr>
<td>ave()</td>
<td>Average of a field</td>
</tr>
<tr>
<td>rms()</td>
<td>Root mean square of a field</td>
</tr>
<tr>
<td>maxr(),</td>
<td>Maximum of a field, for a selection</td>
</tr>
<tr>
<td>minr(),</td>
<td>Minimum of a field, for a selection</td>
</tr>
<tr>
<td>aver(),</td>
<td>Average of a field, for a selection</td>
</tr>
<tr>
<td>rmsr(),</td>
<td>Root mean square of a field, for a selection</td>
</tr>
<tr>
<td>div()</td>
<td>Divergence of a vector field</td>
</tr>
<tr>
<td>rot()</td>
<td>Curl of a vector field</td>
</tr>
<tr>
<td>grad()</td>
<td>Gradient of a scalar/vector field</td>
</tr>
<tr>
<td>ii()</td>
<td>Second invariant of a vector field</td>
</tr>
</tbody>
</table>

Table B.2: Meaning of functions that can be used in expressions.
Appendix C

C-macros for icon modelling

This appendix contains a description of the C-macros that were developed for the design of icons, and the binding of attribute sets to these icons. The macros can be split in five categories:

- attribute binding macros,
- 2D contour curve definition macros,
- 3D geometry definition macros,
- 3D geometry transformation macros,
- administration macros.

The macros of each of these categories are listed below, followed by the code of some icons that are described in Chapter 5.

C.1 Attribute binding

For binding, it is assumed that the attributes are available as elements of an array $a$, thus the elements of an attributeset $A = a_0, \ldots, a_n$ are $a[0], a[1], \ldots, a[n]$. Furthermore, for the definition of parametric functions, one can use to parameter $t$ in the binding macros. As we use C-macros for our modelling language, standard C-expressions and C-functions can be used, as well as the C-functions defined in the mathematical library (include <math.h>).

Examples of expressions that can be used in binding macros are:

\[
\begin{align*}
\sin(a[0]) & \quad \sin a_0 \\
\text{atan2}(a[1],a[2]) & \quad \arctan \frac{a_1}{a_2} \\
a[0] > a[1] & \quad -1 : 1 \quad -1 \text{ if } a_0 \text{ is greater than } a_1 \text{ else } 1
\end{align*}
\]
APPENDIX C. C-MACROS FOR ICON MODELLING

The following operators can be used to bind attribute values to objects:

- **Bind.XY(pos2d,x,y)** binds the 2D position \((x,y)\) to \(pos2d\). \(x\) and \(y\) are functions of the attributes set \(a_0, \ldots, a_n\). **Bind.XY** can also be used to define a parametric function, if the parameter \(t\) is used. E.g. a circle is defined with **Bind.XY(Circle, cos(t), sin(t)).**

- **Bind.Scale(scale,sx,sy,sz)** binds three scale factors \(s_x\), \(s_y\) and \(s_z\) to \(scale\).

- **Bind.Rot(rotation,α,β,γ)** binds three rotation angles (\(α\) around \(x\)-axis, \(β\) around \(y\)-axis, \(γ\) around \(z\)-axis) to \(rotation\). Positive angles are defined according to the right hand rule.

- **Bind.Trans(translation,tr_x,tr_y,tr_z)** binds the translation vector \((tr_x, tr_y, tr_z)\) to \(translation\).

- **Bind.Swp(sweep,f_x,f_y,f_z,sx,sy,sz,α)** binds six functions that determine a general sweep to \(sweep\). The sweep trajectory is determined by \(f(λ) = (f_x(λ), f_y(λ), f_z(λ))\), \(s_x\) and \(s_y\) are scale factors for the contour that is being swept and \(α\) determines the rotation along the trajectory. If these functions do not depend on \(λ\), they are constant along the trajectory.

- **Bind.S(scalar,s)** binds a scalar function \(s\) to \(scale\).

- **Bind.Lin(lindef,a,b)** binds two number \(a\) and \(b\) to the linear deformation \(lindef\). With \(lindef\), a contour can be deformed in the \(x\)-direction with a linear function (see also **XYZ.LinD**).

- **Bind.Quad(quadeff,a,b,c)** binds three numbers \(a\), \(b\) and \(c\) to the quadratic deformation \(quadeff\). With \(quadeff\), a contour can be deformed in the \(x\)-direction with a quadratic function (see also **XYZ.QuadD**).

- **Bind.RGB(rgb,r,g,b)** binds three colour components red, green and blue \((r,g,b)\) to the colour \(rgb\). The value of a colour component should have a value in the interval \([0 \ldots 255]\).

- **Bind.Cmap(cmap,i,n)** binds an index \(i\) to a colour map. The AVS module that implements the icon generator may have up to four colour maps. \(n\) can be used to choose one of these colour maps \((0,1,2,3)\). With the index \(i\) a colour can be determined. As AVS allows for scaling the range for the colour map, there are no limits for the index \(i\).

### C.2 2D contour curve definition

Using the bindings of the previous section, a contour can be defined with the following macros:
• $\text{XY\_SetPen}(\text{pos2d,rgb})$ sets the pen with colour or colour map $\text{rgb}$ at position $\text{pos2d}$ in the $xy$-plane.

• $\text{XY\_LineTo}(\text{pos2d,rgb})$ draws a line to $\text{pos2d}$ in colour $\text{rgb}$.

• $\text{XY\_LineToIC}(\text{pos2d,rgb})$ draws a line to $\text{pos2d}$, where the colour on the line is interpolated from the colour at the starting position to the colour at the end position ($\text{rgb}$).

• $\text{XY\_DrawFunc}(\text{pos2d,number,t_s,t_e,colour})$ draws the function $\text{pos2d}$ for $t = t_s$ to $t_e$, in $\text{number}$ steps.

C.3 From 2D contour to 3D geometry

Contours that have been defined with the 2D contour curve macros can be transformed to 3D geometries by a sweep operation: a rotation sweep around the $x$ or $y$-axis, a translation sweep along the $z$-axis or a sweep along an arbitrary trajectory:

• $\text{Sweep\_R\_X}(\text{number})$ sweeps the previously defined contour around the $x$-axis in $\text{number}$ steps,

• $\text{Sweep\_R\_Y}(\text{number})$ sweeps the previously defined contour around the $y$-axis in $\text{number}$ steps,

• $\text{Sweep\_T}(\text{scalar,rgb})$ sweeps the previously defined contour along the $z$-axis. The length of the sweep is determined by $\text{scalar}$, while the colour is contained in $\text{rgb}$

• $\text{Sweep\_R\_Y}(\text{number})$ sweeps the previously defined contour around the $y$-axis in $\text{number}$ steps,

• $\text{Sweep\_F}(\text{sweep, number,t_s,t_e,rgb})$ generates a generalized cylinder along the trajectory specified by $\text{sweep}$, for $t$ ranging from $t_s$ to $t_e$ in $\text{number}$ steps. The fronts of the cylinder get the colour $\text{rgb}$, the 2D contour already has a colour. $\text{sweep}$ also contains parametric profile curves to scale the 2D contour. For this scaling, two local coordinate frames can be used: either a Frenet frame, or a frame that is incrementally constructed (similar to that described by Klok [43]).

C.4 3D transformations

3D geometries can be transformed with the following macros:

• $\text{XYZ\_Scale}(\text{scale})$ scales a 3D geometry according to the values that are bound to $\text{scale}$.
APPENDIX C. C-MACROS FOR ICON MODELLING

- **XYZ_Rotate(rotation)** rotates a 3D geometry according to the values that are bound to *rotation*.

- **XYZ_TRANSLATE(trans)** translates a 3D geometry according to the values that are bound to *trans*. Before this translation is performed, a global scaling is applied. This scaling is implemented to allow a user to scale the icon from the user interface. To prevent this scaling before the translation (e.g. to move sub-geometries), the next macro can be used.

- **XYZ_TRANS_Ns(trans)** translates according to *trans*, but without the global scalings.

- **XYZ_QuadD(quadef)** performs a quadratic deformation in the *x*-direction on a 3D geometry, according to the following formula (see also Bind_Quad):

\[
\begin{align*}
    x' &= x + ay^2 + byz + cz^2 \\
    y' &= y \\
    z' &= z
\end{align*}
\] (C.1)

- **XYZ_LinD(lindef)** performs a linear deformation in the *x*-direction on a 3D geometry, according to the following formula (see also Bind_Lin):

\[
\begin{align*}
    x' &= x + ay + bz \\
    y' &= y \\
    z' &= z
\end{align*}
\] (C.2)

- **Lock_To_Viewpoint** locks the orientation of the icon in the direction of the viewpoint. If the viewing direction is changed in the AVS Geometry Viewer, the icons stay oriented towards the viewing position.

C.5 Icon administration

There are some extra macros for icon administration:

- **Set_AV(string)** sets the default attribute set for an icon.

- **Set_Type(string)** sets the type of the icon (as it appears in the user interface of the icon module).

- **Set_Name(string)** sets the name of the icon (as it appears in the user interface of the icon module).

- **Write** writes a complete icon to the geometry viewer.

- **EndOfAll** says that no more icons will follow.
C.6 Code for some example icons

The macros for the arrow icon (see Figure 5.5 on page 103) are:

```c
/* * Binding of attributes */
Bind_Scale (ArrScaleX, sqrt(a[3]*a[3]*a[4]*a[4]*a[5]*a[5]), 1, 1)
Bind_XY(ArrA1, 1, 0)
Bind_XY(ArrA2, 0.8, 0.04)
Bind_XY(ArrA3, 0.82, 0.02)
Bind_XY(ArrA4, 0, 0.03)
Bind_XY(ArrA5, 0, 0)

/* * Name of icon */
Set_Type ("Arrow"),
Set_Name ("Arrow 1"),

/* * Default attribute string */
Set_AV ("0 0 0 1 1 1"),

/* * 2D contour definition */
XY_SetPen (ArrA1,YELLOW),
XY_LineTo (ArrA2,YELLOW),
XY_LineTo (ArrA3,YELLOW),
XY_LineTo (ArrA4,CYAN),
XY_LineTo (ArrA5,CYAN),

/* * 3D object by rotation sweep */
Sweep_R_X (15),

/* * 3D transformation */
XYZ_Scale (ArrScaleX),
XYZ_Rotate (AnglesFromVec_3_4_5),
XYZ_Translate (T012),

/* * Write object to geometry viewer */
Write,
```

The colours YELLOW and CYAN are predefined, as is AnglesFromVec_3_4_5, which determines the angles for rotation from the vector direction.

The macros for an ellipse icon (see Figure 5.6 on page 103) are:

```c
/* * Binding of attributes */
Bind_XY(EllA, 0.05*cos(t), 1+0.05*sin(t))

/* * Name of icon */
Set_Type ("Ellipsoid"),
Set_Name ("Ellipsoid 1"),

/* * Default attribute string */
Set_AV ("0 0 0 1.2 1.4 1.6 -0.4 -0.4 -0.4"),
```
APPENDIX C. C-MACROS FOR ICON MODELLING

/* * 2D contour definition */
XY_DrawFunc (CIRCLE,20,0,PI,YELLOW),

/* * 3D object by rotation sweep */
Sweep_R_X (HiRes),

/* * 3D transformation */
XYZ_Scale (S345),
XYZ_Rotate (R878),
XYZ_Translate (T012),

/* * Write object to geometry viewer */
Write,

The macros to generate an Hermite interpolated tube icon (see Figure 5.9 on page 106, with a circular cross section, and without candy stripes) are:

/* * Binding of attributes */

/* * Name of icon */
Set_Type ("Hermite Position+scalar"),
Set_Name ("Hermite 1"),

/* * Default attribute string */
Set_AV ("0 0 0 0.1 3 0 0 0 1 1 0 0.1 2 1 4 0"),

/* * 2D contour definition */
XY_DrawFunc (CIRCLE,10,0,2*PI,MAGENTA),

/* * 3D object by rotation sweep */
Sweep_F (Hermite,5,0,1,MAGENTA),

/* * Write object to geometry viewer */
Write,
Appendix D

Datasets

This appendix contains a short description of the datasets that were used in this thesis. For each dataset, the grid size, data, background and origin are given, and the chapters in which the dataset is used are specified.

D.1 Blunt fin

Grid $40 \times 32 \times 32$, curvilinear

Data density, momentum, stagnation

Background Flow around a blunt fin

Origin Provided with AVS

Remarks Used for iconic visualization example

D.2 Hydrogen

Grid $64 \times 64 \times 64$, Cartesian

Data scalar field

Origin Provided with AVS

Remarks Used in some examples in Chapter 2

D.3 Backward facing step

Grid $25 \times 37 \times 9$, curvilinear
APPENDIX D. DATASETS

Data pressure, velocity

Background Flow in a backward facing step

Origin Courtesy Numerical Mathematics Department, Delft University of Technology

Remarks Used in every chapter

D.4 Turbulent pipe flow

Grid 26 × 11 × 51, curvilinear

Data velocity

Background Direct numerical simulation of turbulent flow in a pipe

Origin Courtesy Laboratory of Aero- and Hydrodynamics, Delft University of Technology (Jaap den Toonder)

Remarks Used in Chapter 6, only a small part of the original dataset is used (original size of dataset: 96 × 128 × 256)

D.5 Flow with chemical reaction

Grid 42 × 42 × 42, Cartesian

Data velocity, concentration of O₂, NO

Background Large-eddy simulation of atmospheric flow with a chemical reaction

Origin Courtesy Laboratory of Aero- and Hydrodynamics, Delft University of Technology (Ivo Bouwman / John Meeder)

Remarks Used in Chapter 6

D.6 Plume flow

Grid 100 × 100 × 100, Cartesian

Data velocity, concentration field

Background Large-eddy simulation of atmospheric flow with a plume

Origin Courtesy Laboratory of Aero- and Hydrodynamics, Delft University of Technology (John Meeder / Ivo Bouwman)

Remarks Used in Chapter 6
Appendix E

Colour plates

This appendix contains the colour plates of this thesis (see also the colour plates on the front and back cover). The next two pages contain images for the turbulent pipe flow case study. These pages are followed by two empty pages, which in turn are followed by the images of the cases with a chemical reaction in a flow field, and a plume flow field.
Colour plate 1: Combined visualizations of $II_D$, $R_2$ and $R$: upper image shows $II_D$ at regions where $|R_2| > 1$; lower image combines this with isosurfaces for $|R|$ at 80. It is clear that regions with a large value for $R$ correspond with the regions with a large value for $II_D$ and $R_2$. (See also Figure 6.5 on page 117.)
Colour plate 2: Visualizations for comparing several quantities with $|R|$, left without an isosurface for $|R|$, right with an isosurface for $|R|$ at 80. Upper row: regions where $\omega \cdot D\omega$ is greater than 2e6; middle row: regions where $\frac{\omega \cdot D\omega}{|\omega||D\omega|}$ is greater than 0.95; bottom row: regions where $\frac{\omega \cdot D\omega}{|\omega|^2}$ is greater than 27. (See also Figure 6.6 on page 118.)
APPENDIX E. COLOUR PLATES

Colour plate 3: Visualization of velocity where reaction speed is high with arrows in coloured region. (See also Figure 6.8a on page 121.)

Colour plate 4: Visualization of $K$ throughout the plume, using coloured cross-markers. (See also Figure 6.10 on page 123.)
Colour plate 5: Visualization of dotprod throughout the plume at nodes where the magnitude of t.mf or cg is relatively large. (See also Figure 6.12 on page 126.)

Colour plate 6: Three coloured 3D Chernoff faces. (See also Figure 5.1 on page 93.)
Summary

This thesis discusses new techniques for scientific visualization. Techniques for the calculation of streamlines on curvilinear grids are investigated. Furthermore, an approach to selective and iconic visualization is discussed.

Chapter 2 gives an overview of existing visualization techniques, focused on techniques for the visualization of fluid flow data. It discusses techniques for each phase of the visualization process. Much attention is given to the visualization mapping phase: the transformation of numerical data to visual primitives. Based on this overview, the goals for this project are determined.

Chapter 3 discusses how streamlines can be calculated on curvilinear grids. Both interpolation and point location are non-trivial problems on curvilinear grids. The transformation of the curvilinear grid to a Cartesian grid appears to give inaccurate results for streamline integration. Three techniques are described that operate directly on the curvilinear grid: Stencil Walk (an iterative technique), approximation of cell boundaries with bilinear patches, and tetrahedrization of cells (geometric approaches). It appears that Stencil Walk is very accurate, but also rather slow. A reasonable alternative is tetrahedrization, less accurate but faster. Furthermore, the relation between accuracy of second order Runge Kutta integration and the accuracy of interpolation, is investigated.

Chapter 4 describes the selective visualization approach. Because datasets contain too much data to be visualized all at once, techniques are necessary that allow to visualize only a selected part of the data. The integration of selective visualization in the general visualization model is discussed. A new selection technique is described, with which content-based selections can be created, and it is shown how selected data can be used in the visualization mapping.

The selection technique is based on "selection expressions": Boolean expressions that denote which data is interesting. These selection expressions are evaluated for each grid node of a dataset, resulting in a set of selected grid nodes. These grid nodes can be used in the visualization mapping in three different ways: they can be visualized directly, they can be used to and highlight clip visualization primitives, and they can be used to automatically determine visualization parameters.

In Chapter 5 iconic visualization is discussed. Iconic visualization is an extension of selective visualization: instead of using selected grid nodes separately,
APPENDIX E. COLOUR PLATES

groups of selected grid nodes are considered to be features in the dataset, and processed as features in the visualization process. Eventually, such a feature can be visualized with one parametric object (an icon).

Each feature is described by a set of attributes. Volume integrals are used for the calculation of attributes. With volume integrals, attributes such as average value, variance and covariance are easily calculated.

A simple modelling language has been developed for the design of icons. With this language, 2D contours can be described. 3D shapes are created from these 2D contours by sweeping. The 3D object can be transformed (scaled, rotated, translated). In each of these three steps, a relation can be established between the icon's parameters and feature attributes.

Chapter 6 contains four case studies, that were used to show the effectiveness the presented techniques. Selective and iconic visualization techniques were applied for each case, to create a visualization that gives an answer to a research question with respect to the dataset. The datasets used are a numerical simulation of a flow in a backward facing step, a direct numerical simulation of a turbulent pipe flow, a large-eddy simulation of a flow with a chemical reaction and a large-eddy simulation of a plume flow. In each of the cases effective visualizations, were created efficiently. With these visualizations, the research questions were answered easily.
Samenvatting

In dit proefschrift wordt een aantal nieuwe technieken voor wetenschappelijke visualisatie beschreven. Er wordt ingegaan op technieken voor het bepalen van stroomlijnen op kromlijnige roosters. Daarnaast wordt een selectieve en iconische visualisatie aanpak beschreven.

Hoofdstuk 2 geeft een overzicht van bestaande visualisatie technieken, met name gericht op de visualisatie van stromingsdata. Er wordt beschreven welke technieken voor de verschillende stadia in het visualisatie proces beschikbaar zijn. De nadruk ligt hierbij op de visualisatie mapping: de transformatie van numerieke data naar visuele primitieven. Aan de hand van het overzicht worden de doelen voor het onderzoek bepaald.

Hoofdstuk 3 bespreek hoe stroomlijnen voor stromingsdata op kromlijnige roosters bepaald kunnen worden. Zowel interpolatie als het vinden in welke rooster-tercel een bepaald punt zich bevindt (puntlocatie) is voor kromlijnige roosters niet eenvoudig. Een transformatie van het kromlijnige rooster naar een Cartesian rooster, een in de numerieke stromingsleer veel gebruikte techniek, blijkt onnauwkeurige resultaten te geven. Vervolgens wordt een drietal technieken besproken die rechtstreeks werken op het kromlijnige rooster: Stencil Walk (een iteratieve techniek), benadering van de celwanden met bilineaire patches en tetraëdrisatie van cellen (geometrische technieken). Van deze technieken blijkt de Stencil Walk erg nauwkeurig, maar ook erg traag. Een redelijk alternatief is de opdeling van het rooster in tetraëders, minder nauwkeurig maar sneller. Vervolgens wordt de relatie onderzocht tussen de nauwkeurigheid van tweede orde Runge Kutta integratie en de nauwkeurigheid van interpolatie.

Hoofdstuk 4 beschrijft de selectieve visualisatie aanpak. Omdat datasets tegenwoordig te veel data bevatten om alle data tegelijk af te beelden, zijn technieken nodig die de mogelijkheid bieden om een deel van de data te selecteren en af te beelden. Er wordt besproken hoe selectieve visualisatie in het algemene visualisatie-model kan worden opgenomen. Vervolgens wordt een nieuwe selectie techniek beschreven, waarmee data op basis van inhoudelijke criteria geselecteerd kan worden, en wordt getoond hoe geselecteerde data gebruikt kan worden in de visualisatie mapping.

De selectie techniek is gebaseerd op “selectie-expressies”: Booleaanse expressies die aangeven welke data van belang zijn. Deze selectie-expressies worden
voor elk roosterpunt van een dataset geëvalueerd, wat resulteert in een verzameling roosterpunten. Deze roosterpunten kunnen op drie manieren in de visualisatie mapping gebruikt worden: ze kunnen direct gevisualiseerd worden, ze kunnen gebruikt worden om visualisatie-primitieven te clippen of te benadrukken, en ze kunnen gebruikt worden om visualisatie parameters automatisch te bepalen.

In Hoofdstuk 5 wordt iconische visualisatie beschreven. Iconische visualisatie is het logisch vervolg op selectieve visualisatie: in plaats van de geselecteerde roosterpunten rechtstreeks te gebruiken, worden groepen van geselecteerde roosterpunten als eenheden ("features") gezien, en als eenheden verwerkt in het verdere visualisatie proces. Uiteindelijk kunnen deze eenheden dan met één parametrisch object, een icon, afgebeeld worden.

De eenheden kunnen beschreven worden door een verzameling kenmerken: de attributen. Voor het bepalen van attributen worden volume integralen gebruikt. Hiermee zijn allerlei kenmerkende waarden, zoals zwaartepunt, gemiddelde waarde, variantie en covariantie over het geselecteerde gebied, eenvoudig te bepalen.

Voor het ontwerpen van iconen is een eenvoudige modelleertaal ontworpen. Met deze taal kunnen 2D contouren beschreven worden. Van een 2D contour kan met een sweep-operatie een 3D object gemaakt worden. Dit 3D object kan vervolgens getransformeerd (geschaald, geroteerd en getransleerd) worden. In elk van deze drie stappen kunnen parameters van het icon gekoppeld worden aan attributen van een geselecteerd gebied.

Hoofdstuk 6 bevat tenslotte vier case studies, met behulp waarvan de gepresenteerde technieken getest zijn. Voor elke case worden selectieve en iconische visualisatie technieken toegepast, om een visualisatie te creëren die antwoord geeft op de bij de dataset behorende vraagstelling. De gebruikte datasets zijn een numerieke simulatie van stroming in een 'backward facing step', een directe numerieke simulatie van een turbulente pijpstroming, een large-eddy simulatie van een stroming met een chemische reactie en een large-eddy simulatie van een stroming met een pluim. In elk van de cases was op een eenvoudige en efficiënte manier een effectieve visualisatie te maken waarmee het antwoord op de vraagstelling gevonden kon worden.
Curriculum Vitae

