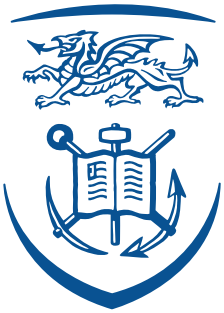
Ph.D Thesis

Stream Surface Seeding for Flow Visualisation

Matthew Edmunds
496425

March 2014



Swansea University
Prifysgol Abertawe

Thesis submitted to Swansea University for completion of a Doctoral in Computer Science.

Department of Computer Science, Swansea University
Singleton Park, Swansea SA2 8PP

Abstract

WITH increasing computing power, it is possible to process more complex fluid simulations. However, a gap between increasing data size and our ability to visualise them still remains. Despite the great amount of progress that has been made in the field of flow visualisation over the last two decades, a number of challenges remain. Difficulties can stem from attempting to capture all flow features, the speed of computation, and spatial perception. This effects the ability of the domain engineer to study, capture, and visualise, 3D flow phenomena.

A review of 3D flow visualisation literature revealed little presented work for stream surface placement and visualisation methods. Stream surfaces are a useful tool for visualising 3D flow due to their ability to convey different attributes from their structure. It is important that the domain engineer can guide the placement, interact with, and examine specific properties of flow data. Streamlines are one standard tool domain engineers use for visualising flow data. Although a variety of automatic seeding approaches have been proposed for streamlines, little work has been presented for stream surfaces. Previous research generally focuses on manual placement of stream surfaces. Little attention has been given to the problem of automated stream surface seeding. The placement or seeding of stream surfaces in 3D flow fields face a number of challenges. These challenges include perception, occlusion, and the appropriate representation of flow characteristics.

Flow visualisation with a focus on stream surface techniques forms the basis of this thesis. We detail our investigation into new algorithms with discussions of their applicability and their relative strengths and drawbacks. Our goal is to present domain engineers with state of the art techniques furthering their ability effectively and efficiently study, capture, and visualise, 3D flow phenomena.

We first describe an algorithm for defining seeding curves at the domain boundaries from isolines generated from a derived scalar field. We detail the generation of stream surfaces integrated through the flow and discuss the associated challenges of surface termination and occlusion. Extending this work we present stream surface seeding and filtering and discuss a technique for effective surface termination. We present an algorithm that automatically seeds new interior surfaces, to represent locations not captured

by the boundary seeding. Our approach is designed to capture the flow characteristics utilising illustrative techniques alleviating occlusion, and providing options for filtering.

Our next approach utilises clustering techniques to simplify the vector field enabling a range of abstractions for the density and placement of seeding curves and their associated stream surfaces. It is important that the domain engineer can define and target particular characteristics of the flow. In combination with improved placement we further illustrative techniques for better perception. Continuing towards our goal of providing effective techniques for use by domain engineers, we then tailor this algorithm towards a specific application: The Bloodhound project. The Bloodhound project develops a rocket propelled land vehicle designed to break the land speed record. We describe modifications to the seeding algorithm for large CFD simulation data including: searching large unstructured grids, reducing the memory footprint of the algorithm data structures, and customisation of the distance metric used to cluster the simulation data.

Declaration Statements

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed .....(candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extent and nature of the correction is clearly marked in a footnote(s).

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .....(candidate)

Date .....

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for interlibrary loan, and for the title and summary to be made available to outside organisations.

Signed .....(candidate)

Date .....

Acknowledgements

I WOULD not have been able to finish this thesis without the support of a number of people who have had a direct or indirect impact on my three years of study for a PhD at Swansea University. Having a retrospective view on this long journey, I'm very thankful to all these people.

I would like to give thanks to my supervisor Dr. Robert S. Laramee during my three years of study. I am very thankful to the Department of Computer Science for providing me with this opportunity. I also want to give special thanks to; Professor Min Chen for invaluable advice prior to studying for a PhD, Dr Matt Jones for many insightful chats by the coffee machine, and Dr Mark Jones for his technical advice. I also would like to thank some research colleagues who have been a pleasure to work with: Ian Doidge, Phillip James, Matthew Gwyne, Fredrik Norvsberg, Zhenmin Peng, Zhao Geng, Liam O'Reilly, David Chung, Ben Spencer, and Andy Gimblet. Special thanks go to Phillip James for proof reading this thesis.

I also want to thank co-authors of our published research papers: Dr. Guoning Chen, Dr. Eugene Zhang, Dr. Nelson Max, Dr. Ben Evans, Dr. Ian Masters, Dr. Colin Ware and Dr. Nick Croft. This thesis wouldn't have come to fruition, without the support of these people who directly or indirectly influenced me during my three years of PhD study. I'm very thankful to all these people.

My final words are dedicated to my family for their endless and tireless guidance, support, care, patience and encouragement during my study. Thank You.

Publications

THE list of papers published while researching the subject of stream surface placement are listed in this section. The following publications form the base from which this thesis is written:

- Matt Edmunds, Robert S. Laramee, Guoning Chen, Nelson Max, Eugene Zhang, and Colin Ware. **Surface-Based Flow Visualization**. In Computers & Graphics, Volume 36, Issue 8, December 2012, Pages 974 to 990.
- Matt Edmunds, Tony McLoughlin, Robert S. Laramee, Guoning Chen, Eugene Zhang, and Nelson Max. **Automatic Stream Surfaces Seeding**. In Eurographics 2011 Short Papers, Pages 53 to 56, Llandudno, Wales, UK, April 11 to 15 2011.
- Matt Edmunds, Robert S. Laramee, Guoning Chen, Eugene Zhang, and Nelson Max. **Advanced, Automatic Stream Surface Seeding and Filtering**. In Theory and Practice of Computer Graphics, September 2012, Pages 53 to 60.
- Matt Edmunds, Robert S. Laramee, Rami Malki, Ian Masters, Nick Croft, Guoning Chen, and Eugene Zhang. **Automatic Stream Surface Seeding: A Feature Centered Approach**. In Computer Graphics Forum, Volume 31, Issue 3, Part 2, June 2012, Pages 1095 to 1104.
- Matt Edmunds, Robert S. Laramee, Ben Evans, Guoning Chen. **Stream Surface Placement for a Land Speed Record Vehicle**. Technical Report, March 2013.
- Matt Edmunds, Robert S. Laramee. **Design of a Flow Visualisation Framework**. Technical Report, June 2013.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

DATA visualisation is the study of mapping abstract data into a visual form that makes data more humanly digestible, and to improve cognitive understanding. The abstract data can be both numerical and non numerical. Examples of non numerical data are present in literature from geographic, social, environmental, and cultural domains. The requirement to efficiently and effectively extract implicit, non trivial, unknown and potentially useful patterns and knowledge from data has roots dating back prior to recorded history. One such example is the Aboriginal rock art in Western Aus-

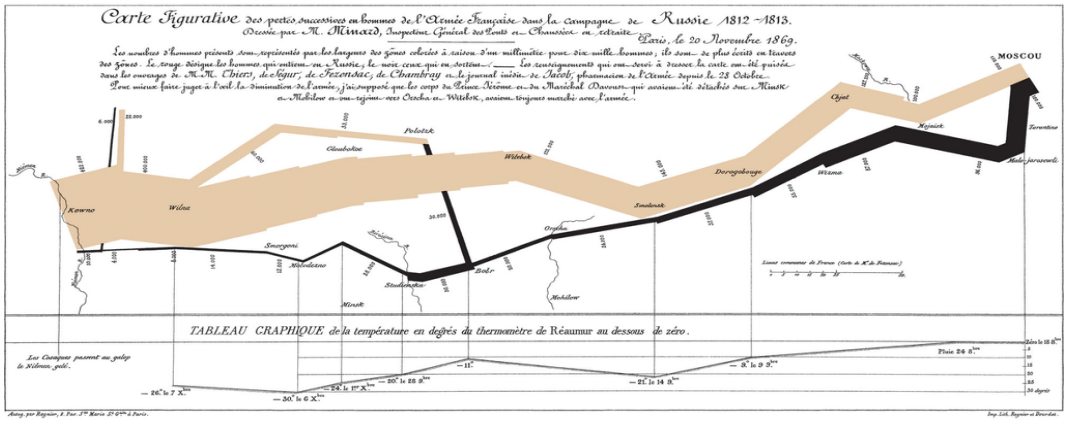


Figure 1.1: *This graph shows the size of Napoleon's army mapped to the width of the path across the map. Its shows the outward and return journeys. Image courtesy of [Fri08].*

tralia's Pilbara region, and the Olary district of South Australia, estimated to be up to around 40,000 years old [Abo]. Rock art gives us descriptive information about social activities, material culture, economy, environmental change, myth and religion.

In more recent times the French engineer Charles Minard (1781-1870) illustrated in Figure 1.1 the disastrous result of Napoleon's failed Russian campaign of 1812 [Mer]. This statistical graphic provides valuable insight into the factors contributing to the failure of this campaign. Later in the same century (1820-1910) Florence Nightingale's place in history is at least partly linked to her use of graphical methods in Figure 1.2 to convey complex statistical information dramatically to a broad audience [Lyn]. After witnessing deplorable sanitary conditions in the Crimea, she wrote an influential text including several graphs which she called "Coxcombs". The work covered matters affecting the health, efficiency and hospital administration of the British Army.

The introduction of computers during the 20th century brought the ability to compute, process and store large amounts of data. As with data which precedes the computer age, we use visualisation techniques to efficiently extract knowledge, perceive patterns, and gain useful insight into the data. However, as the quantity and complexity of the data increases, our ability to efficiently and effectively deduce knowledge from it decreases. Research into visualisation techniques suitable for representing complex data is becoming an increasingly important focus. With the advancement of technological media and computational power becoming more accessible, many applications such as simulation, data logging, business and financial data are growing both in complexity (number of

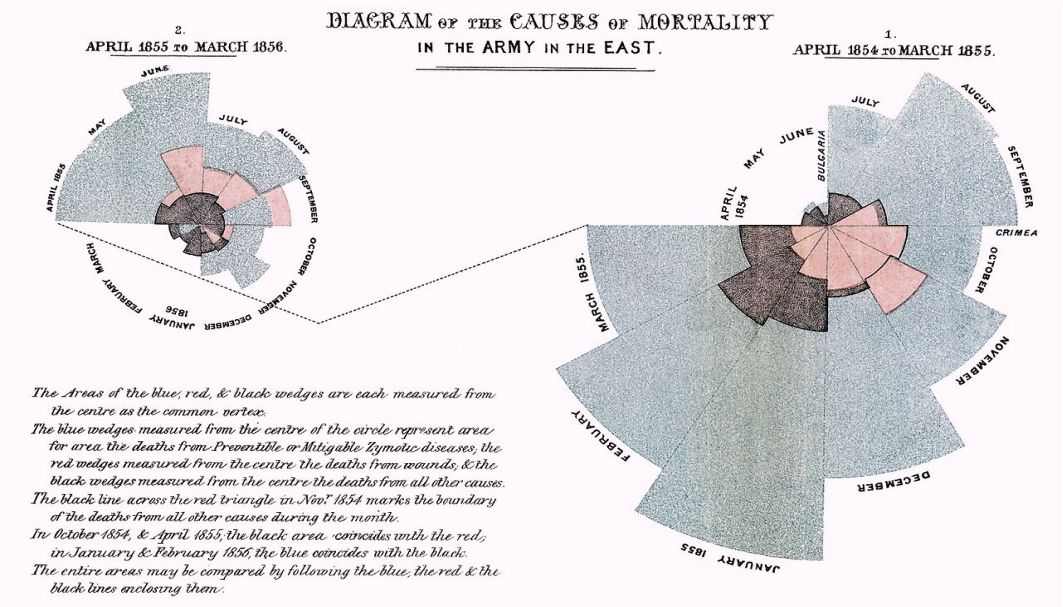


Figure 1.2: *The inset text describes what the contributing factors where to the causes of death on the battle field for the British Army. The graphics visually represent the data supporting the text. [Fri08]*

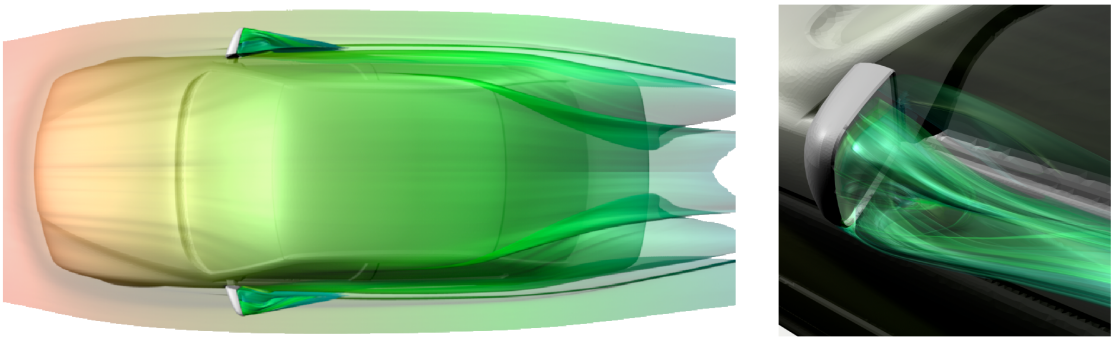


Figure 1.3: *This figure shows stream surfaces in an automotive application. This technique, by Garth et al. [GKT*08], enhances the perception of the flow structures over the vehicle bodywork.*

attributes) and quantity.

Visualisation techniques map raw or derived data to an intuitive visual metaphor. The type of visualisation approach required is dependant on the given data. Typically, visualisation approaches are classified into two general subcategories depending on the properties of data; information visualisation and scientific visualisation [PNB02]. Information visualisation predominantly deals with the visual representation of abstract data which consists of data that is not coupled within a spatial domain. Examples of such data types are commonly found in practice such as finance or business domains. Visualisation approaches such as histograms, pie charts, scatter plots, and parallel coordinates are frequently used for this subset [War04]. Scientific visualisation deals with computational scientific data generated from engineering simulation or modelling data. Some practical domains that use scientific data are medical, flow simulations, and weather ensembles. Common techniques that have been used to visualise such data include glyphs, streamlines, and volume rendering [PNB02].

1.1 Flow Visualisation

As flow data increases in size and complexity, it becomes more important to support effective exploration of the features and characteristics. Many of today's applications for flow visualisation are centred around fields such as aerospace, automotive, energy production, and other scientific research. Automotive design tends to focus on aerodynamic drag to help improve the efficiency of the vehicle. Figures 1.3 and 1.4 demonstrate flow distribution across the vehicle body aiding the engineer in studying drag characteristics. There is also an emphasis on airflow around the engine compartments which provide much needed cooling for the various engine/cockpit heat exchange systems. The heat exchange systems themselves are also an important focus for analysis. Aerospace also focuses on aerodynamics and engine design for aircraft, with similar goals for space craft. Electromagnetism and turbine design within the energy production industries are other common applications. One example application of visualisation within as-

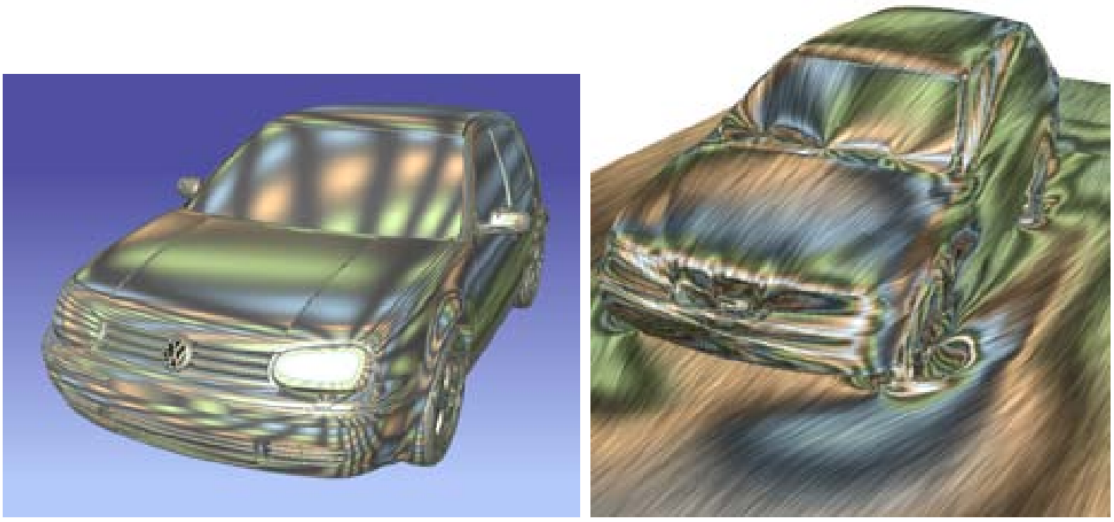


Figure 1.4: *This figure shows hatched cycle shading in an automotive application. This technique, by Weiskopf and Hauser [WH06], enhances the perception of flow over the surface of the vehicles.*

trophysics is the simulation of ionisation front instability [IEE]. Another area of study is the topic of acoustic flow to simulate and visualise such things as engine exhaust, and speaker cabinet design [Tan]. The medical field uses visualisations to study such phenomena as blood flow, for example, when designing heart pumps to support failing hearts. Another area of utilisation is weather systems analysis by organisations such as the MET office [GOV].

There are many different origins of vector data. In general, these are obtained through intricate flow simulation and flow measurement. Flow simulation is often used to predict real world conditions both as an aid to design and for the analysis of large physical systems which may not be captured or recorded with the existing devices. This is especially true for very large and expensive projects such as aircraft, ship, and car design. The cost of building physical prototypes for testing can be great. Therefore, the ability to simulate the test conditions using virtual representations such as CAD (Computer Aided Design) data combined with computational systems such as CFD (Computational Fluid Dynamics) is highly beneficial. CFD is a method in which continuous movement of a fluid is modelled using numerical methods. The three main simulation steps are:

- Preprocessing: Geometry creation, mesh generation, definition of boundary conditions and fluid properties.
- Solving: Performing the computation using numerical methods.
- Post Processing: Visualisation of the CFD results for analysis.

The application of flow visualisation techniques for post processing CFD problems is essential for engineers and practitioners to gain an understanding of the information the

data contains. Current techniques have been integrated into a wide variety of test and simulation systems, for example; Fluent [ANS], EnSight [CEI], Tecplot [Tec]. Allowing the engineers to explore and evaluate data from within these systems in visual a form is key to effectively gain insight. The graphical representation and exploration of data not only allows for much faster analysis but also enables non-experts to understand the underlying phenomenon.

1.2 Basics of Fluid Dynamics and CFD

In the following subsection we will discuss the set of basic fluid equations underpinning the concept of fluid dynamics. These equations are used to model the characteristics of fluid flow using CFD numerical solvers. A moving fluid can be mathematically described by two scalar fields and a vector field. A scalar field is where every discrete point within the spatial domain has a single real value $\mathbb{R}$. A vector field is where every discrete point within the spatial domain has a 3 tuple of real values $\mathbb{R}^3$ representing a vector quantity. The vector field describes the state of fluid velocity $\mathbf{v}$ across the spatial domain, and the temporal domain. The scalar field of any two thermodynamic quantities of the fluid; pressure p and density ρ being the most common, are needed to fully describe the state of the fluid. Other thermodynamic quantities, such as energy, momentum and temperature, are derived directly from the results of the simulation.

Conservation of Mass The continuity equation, [And11] Section 2.4, describes the conservation of mass of a fluid flowing through a fixed volume.

$$\frac{\partial \rho}{\partial t} + \rho(\nabla \cdot \mathbf{v}) + \nabla\rho \cdot \mathbf{v} = 0 \tag{1.1}$$

Where ρ is density, $\mathbf{v}$ is velocity vector, ∇ is the gradient operator, and t is time. For more information on the gradient operator ∇, and other notations, refer to Appendix A.

Incompressible Fluid An incompressible fluid does not change its volume with a change in pressure; it has constant density ρ. Some liquids may be approximated with this concept for example; water, petroleum, mercury. Both the derivative with respect to time and the gradient of the density in Equation 1.1 therefore disappear, [And11] Section 3.6, and [MYO10] Section 6.2. For incompressible flow the continuity equation reduces to:

$$\nabla \cdot \mathbf{v} = 0 \tag{1.2}$$

An incompressible fluid is still bound by the continuity equation e.g. the conservation of mass. Therefore the flow must be divergence free, and cannot contain any sources or sinks.

Inviscid Flow The Euler equation describes the motion of inviscid flow. An inviscid fluid is an ideal fluid of zero viscosity. An inviscid fluid does not exhibit any internal shear stress, for example, a layer of fluid can slide over another layer of fluid and not generate any resisting force due to friction.

The Euler equation together with the Continuity equation 1.1 describes the motion of an inviscid fluid ([MYO10] Section 8.2.2):

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v} = -\frac{1}{\rho} \nabla p + \mathbf{g} \tag{1.3}$$

Where p is pressure, and $\mathbf{g}$ is the gravity vector. This model approximates fluid flow where the effects of viscosity, energy dissipation, and thermal conductivity are considered negligible.

Viscous Flow The Euler model describing fluid flow ignores the effect of internal friction, however, all fluids have some amount of viscosity. For example, one layer of fluid sliding over another layer of fluid now generates a resistive force as a result of friction. This is described as sheer stress. The friction force:

$$\nu \nabla^2 \mathbf{v} \tag{1.4}$$

is proportional to the Laplacian of the velocity ∇^2 and the Kinematic viscosity:

$$\nu = \frac{\mu}{\rho} \tag{1.5}$$

where μ is the dynamic viscosity and ρ is the density, [And11] Section 18.2. Adding the friction force to the Euler equation 1.3, we obtain the Navier Stokes equation:

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{v} + \mathbf{g} \tag{1.6}$$

Together with the conservation of mass Equation 1.1 this equation describes the motion of a viscous incompressible fluid, [MYO10] Section 8.2.2. The book by Hirsch [Hir89] details the discretisation of the above equations and the physical models these equations describe.

Types of CFD Meshes

As described in Section 1.1 one of the inputs to the CFD solver is a mesh describing the discretisation of the spatial domain. This mesh is also used in the post processing step of the pipeline by the visualisation tool visualising the simulation data. The mesh can take a number of different forms. In this section we will discuss the mesh types provided for visualisation in this thesis.

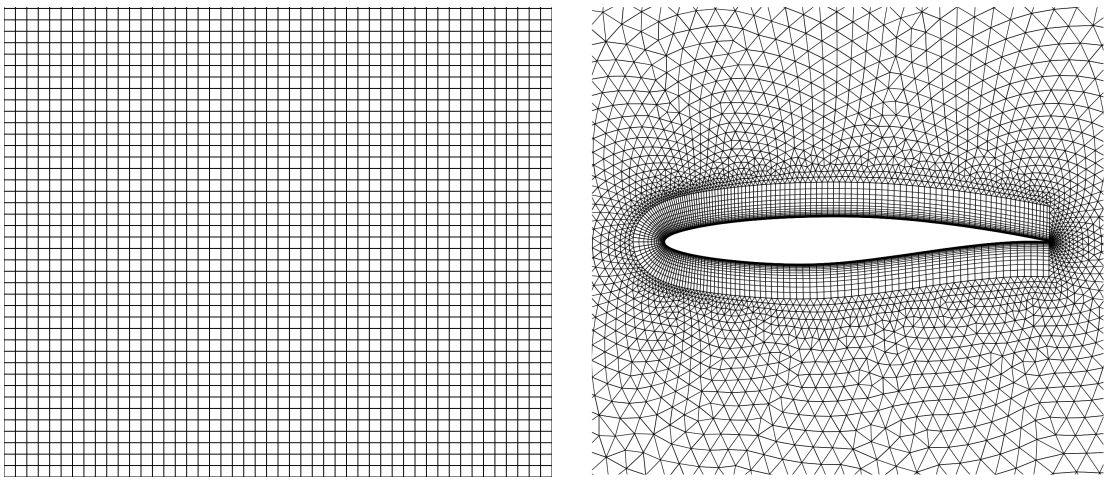


Figure 1.5: *These illustrations demonstrate a two dimensional version of a regular Cartesian grid (left), and an irregular polygonal grid [Sta] (right).*

Meshes are subcategorised into two main types; regular, and unstructured, See Figure 1.5. Regular meshes, or structured grids, are defined by specifying the dimensions in the topological ijk coordinate system. The ijk coordinate system is a parametrisation of the global xyz coordinates. The regular grid is essentially a tessellation of the three parametric dimensions ijk into a Cartesian grid. The advantage of this type of grid is that the coordinates of the vertices are not stored as they can by computed from the linear relationship between the parametrised coordinates and the global coordinates. The parametrised coordinates are directly used to index 3D data arrays storing the CFD solution data. This makes access time very fast and reduces the memory footprint of the data.

Unstructured meshes, or unstructured grids, are a tessellation of the spatial domain into a set of irregular sized polygons usually tetrahedrons or hexahedrons. Along with the vertex data, a connectivity list is also required to describe each of the polygonal cells. Although relative to the regular grid more data is required to be stored, the unstructured grid cells may vary greatly in size. This allows refinement in areas of expected complexity, and coarseness in areas of simplicity. Overall, this can significantly improve the memory usage to represent the mesh. There is a significant overhead, however, when sampling the mesh. Refer to Appendix A for more information on cell structures and interpolation.

2 Flow Visualisation Classification

Flow visualisation is one of the classic subfields of scientific visualisation. The techniques in this field of study can be classified into four general categories: direct, geometric, texture-based and feature-based flow visualisation [PVH*03, LHD*04, LHZP07, Lar08, PL09, MLP*10, ELC*12b]. Each of the following classifications are discussed in greater detail throughout Chapter 2.

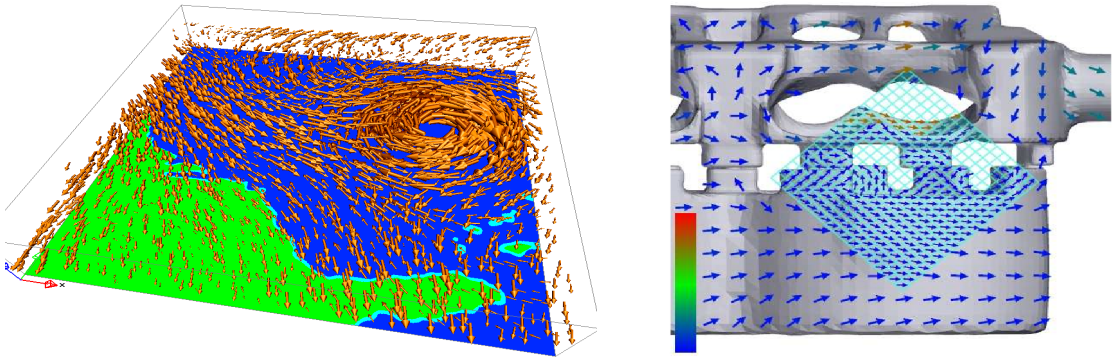


Figure 1.6: *These illustrations demonstrate direct visualisation techniques. Arrow glyphs directly represent the underlying data. Image courtesy of Peng et. al. [PGL*12]*

Direct Flow Visualisation Techniques: Direct visualisation techniques directly map the vector field to primitives. Common approaches visualising flow include placing arrow glyphs at each sample point to depict the associated vector field. These direct techniques are able to make flow visualisation universally and intuitively understandable. It has been used in many applications of CFD flow visualisation. Direct techniques can suffer from a lack of visual coherence and might suffer from visual complexity and occlusion. See Figure 1.6 for examples of direct flow visualisation.

Texture Flow Visualisation Techniques: This approach is rendered with convolved textures to reflect the local properties of the vector field. Texture-based techniques provide a dense and coherent visualisation even in areas of complex flow. It has been successfully applied to 2D and 2.5D (surface) data, but can suffer from visual complexity and occlusion when applied to 3D volumetric flow data. For examples of texture-based techniques, refer to Figure 1.7.

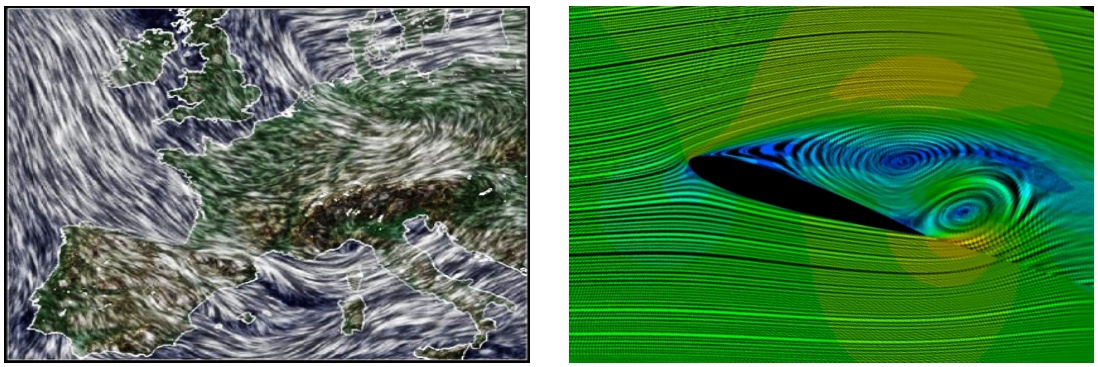


Figure 1.7: *These illustrations demonstrate texture-based visualisation techniques. The pixels of a noise textures are smeared by a small perturbation in the direction of the vector field. Left image courtesy of Laramee et. al. [LEG*08]. Right image courtesy of Shen et. al. [SK98]*

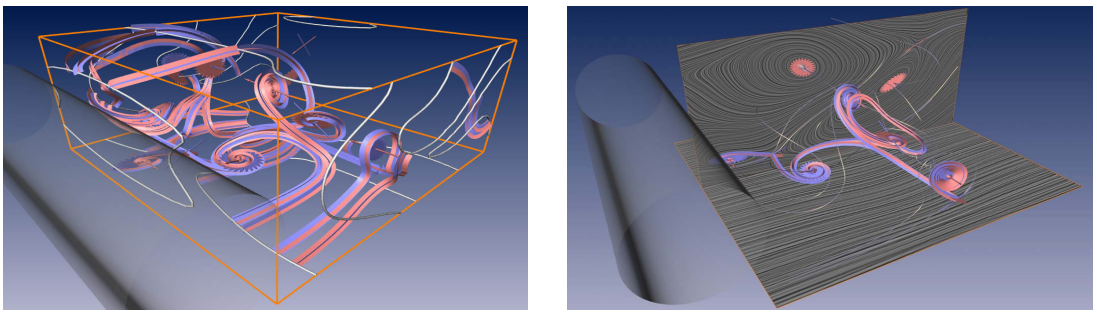


Figure 1.8: *These illustrations demonstrate feature-based visualisation techniques. The images display the topological extraction of flow features. Images courtesy of Theisel and Weinkauf et. al. [TWHS03] and [WTHS04].*

Feature Flow Visualisation Techniques: Feature-based techniques extract subsets of data that are deemed interesting by the user. The visualisation is then based on these extracted subsets rather than the whole dataset. There is significant computational cost when dealing with the complexity of feature extraction, especially when visualising flow based on 3D unstructured CFD meshes. Refer to Figure 1.8 for examples of feature-based flow visualisation.

Geometric Flow Visualisation Techniques: This approach represents flow with geometric primitives. A typical example of geometric techniques is the streamline. Trajectories are computed from an initial location within the domain (seeding point) using integration techniques such as the Runge-Kutta integration scheme (Refer to Appendix A). The resulting geometric objects from these trajectories are then rendered. Due to its comparatively accurate and coherent result, geometric flow visualisation has been widely used to visualise almost all types of CFD data. However, visual clutter and occlusion can stem if poor seeding (placement) strategies are applied to a 3D domain. See Figure 1.9 for examples of streamlines and stream surfaces. Geometric flow visualisa-

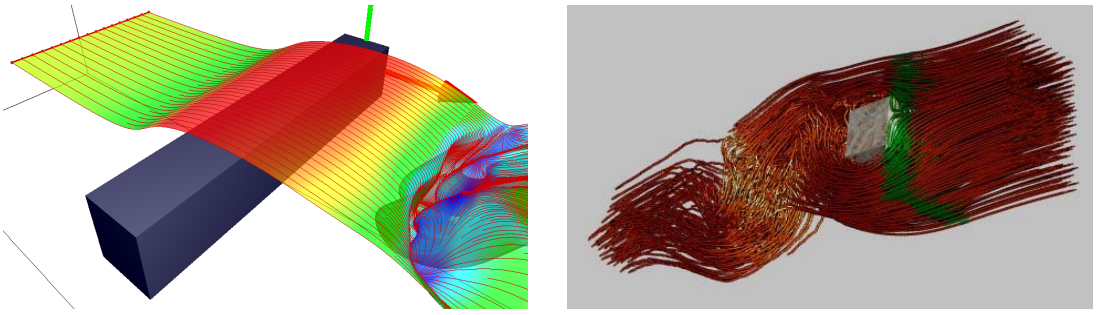


Figure 1.9: *These illustrations demonstrate geometric visualisation techniques. The left image demonstrates streamlines rendered onto a stream surface which is propagated through the velocity field from a seeding curve. The right illustration shows a dense streamline visualisation technique. Image courtesy of Mattausch et. al. [MT*03].*

tion is the context of the research presented in this thesis, with a focus on the effective placement of stream surfaces in a 3D flow field.

3 The Challenge of Surfaces and Placement

Surface-based flow visualisation methods inherit some common problems associated with flow visualisation in general. Examples of these challenges include large, time dependent simulation data requiring the utilisation of out of core techniques, and the handling of unstructured data. Surface-based methods also face their own unique challenges which we discuss in more detail in the following chapters.

Surface construction is a key topic because the surface must represent an accurate approximation of the underlying simulation. When using surfaces the problem of occlusion occurs frequently. This may stem from multiple surfaces that occlude one another, a large surface that produces self occlusion, or a combination of both. While surfaces offer many advantages in terms of perception, a basic visualisation of the surface alone may not provide sufficient information about the underlying data. For example, a stream surface alone does not show the behaviour of inner flow contained within the surface. There is a strong correlation between seeding and occlusion of integral surfaces. Seeding too many surfaces, or seeding them in such a way that they occupy the same region of the domain leads to high levels of occlusion.

Manual placement is the most common stream surface placement method currently used by the CFD engineer. Traditionally, determining the location for seeding of streamlines and surfaces has presented a significant challenge the the engineers studying CFD simulations. The challenges are in terms of the time required by the user to identify the important flow features, and also in terms of the consistency when comparing simulation data for different engineering designs or configurations that have been computed on different meshes. Since aerodynamic phenomena can affect the macroscopic behaviour of the body, often at the very small scale, e.g. flow separation/detachment or vortex shedding, the detection (or non detection) of such features is highly sensitive to the initial seeding of streamlines and stream surfaces. An efficient and robust system for interpreting the vector field resulting from the CFD analysis is therefore highly desirable by the CFD engineer.

4 The Evaluation of a Visualisation Framework

To meet the demands of the CFD engineer, a visualisation framework is developed in collaboration with experts from Swansea University College of Engineering. In this section we discuss aspects of the collaboration pertinent to the evaluation of the proposed algorithms. The evaluation of our algorithms are in the form feedback from the experts based on provided visualisations and collaborative use of the software framework. Our

approach is in part motivated by the need for feedback reviewing the visual performance of the visualisation results, rather than mathematical evaluation of the underlying model. It is the visualisation that seeks to inform the target CFD engineer.

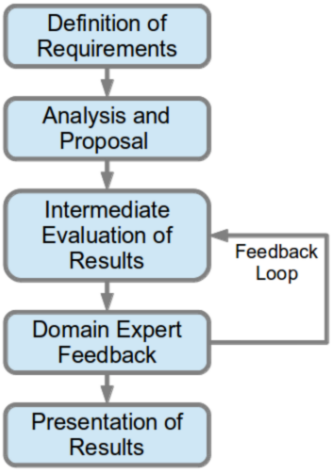


Figure 1.10: *This flow chart illustrates the basic evaluation process undertook while working with the domain experts. During the process a number of feedback iterations may occur prior to the final results being presented.*

As part of our collaboration we worked with a number of experts from Swansea University College of Engineering. The engineering domain experts included; Dr Ian Masters, and Dr Rami Malki of the Marine Energy Research Group (MERG) [Swaa], and Dr Ben Evans of the Bloodhound group [Nob]. The general area of expertise of these domain experts is numerical modelling including CFD.

The approach we took involved a set of meetings and demonstrations that together formed the process shown in Figure 1.10, and the timeline shown in Figure 1.11. The process starts with the aim of discovering, and then defining, what the domain expert actually requires. An analysis of the requirements is then performed; defining any assumptions, specifying the problem logic, and formulating a proposal. This proposal is then implemented, and a set of results is reviewed by the domain experts either interactively or from static visualisations. The domain expert provides feedback which is used to update the model. After refinement of the model is complete, the final results and feedback are generated for inclusion into a formal written document i.e. journal papers and this thesis. Output from the collaboration with MERG resulted in the work described in Chapter 4, and collaboration with the Bloodhound group resulted in the work described in Chapter 5.

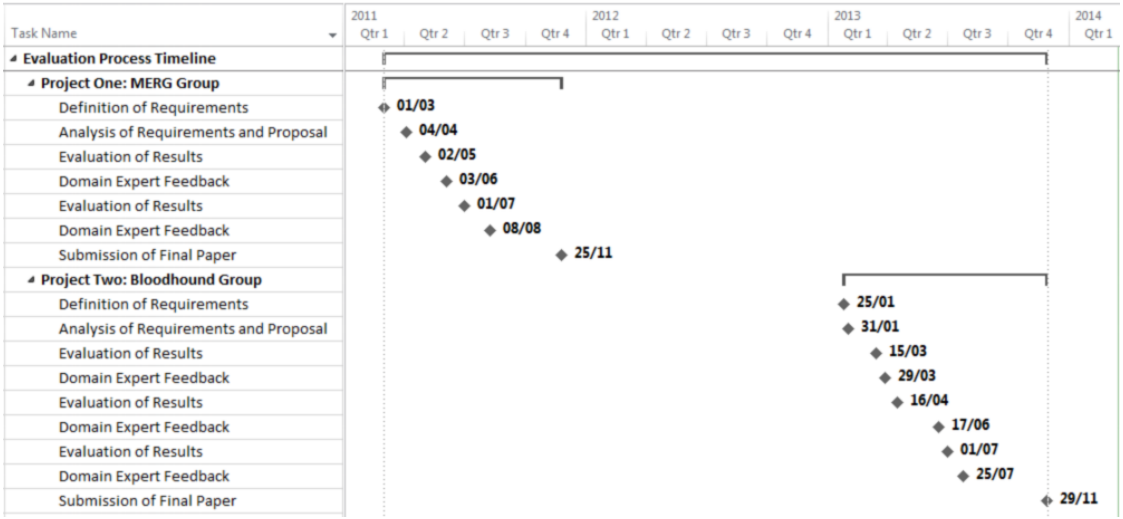


Figure 1.11: *This GANTT chart demonstrates the key events for both projects throughout the evaluation process.*

5 Proposed Solutions: An Overview of Contributions

In this thesis we address the challenges of visualising simulation data with a focus on automated stream surface placement. We apply our techniques to a range of flow data from analytical to large unstructured CFD data. The proposed algorithms are studied and evaluated in conjunction with domain experts who specialise in CFD.

Surface-based Flow Visualisation: Despite the great amount of progress that has been made in the field of flow visualisation over the last two decades, a number of challenges remain. While the visualisation of 2D flow has many good solutions, the visualisation of 3D flow still poses many problems. Flow visualisation with a focus on surface-based techniques forms the basis of the literature review in Chapter 2. The review includes surface construction techniques and visualisation methods applied to surfaces. Detailed is an investigation into these algorithms with discussions of their applicability and their relative strengths and drawbacks. Reviewed are the most important challenges when considering such visualisations. Challenges such as domain coverage, speed of computation, and perception remain key directions for further research [ELC*12b].

Advanced, Automatic Stream Surface Seeding and Filtering: Chapter 3 presents a novel automatic approach to the seeding of stream surfaces in 3D flow fields. Streamlines and stream surfaces are standard tools for visualising 3D flow. Although a variety of automatic seeding approaches have been proposed for streamlines, little work has been presented for stream surfaces. A set of seeding curves are defined and prioritised at

the domain boundaries from isolines computed from a derived scalar field. Surfaces are then traced from the seeding curves through the flow field. An algorithm that automatically seeds new interior surfaces to represent locations not captured by the boundary seeding is presented. The user controls the separation distance from the initial surface set. Discussed is a technique for effective surface termination using a distance field to aid the reduction of visual clutter. We also present the results of this algorithm, how we achieve satisfactory domain coverage, and the capture of flow field features. Strategies for resolving occlusion resulting from seeding multiple surfaces are also presented [EML*11] and [ELC*12a].

Automatic Stream Surface Seeding: A Feature Centred Approach: Chapter 4 introduces a novel automatic stream surface seeding strategy based on vector field clustering. It is important that the user can define and target particular characteristics of the flow when placing surfaces. Our strategy allows the user to specify different vector clustering parameters enabling a range of abstraction for the density and placement of seeding curves and their associated stream surfaces. We demonstrate the effectiveness of this automatic stream surface approach on a range of flow simulations and incorporate illustrative visualisation techniques to assist in analysing the flow. Domain expert evaluation of the results provides valuable insight into the users requirements and effectiveness of our approach [ELM*12].

Stream Surface Seeding for a Land Speed Record Vehicle: Chapter 5 describes a novel cluster-based, automatic stream surface seeding strategy for structured and unstructured CFD data. The algorithm described in Chapter 5 is adapted to tailor it towards a specific application: The Bloodhound SSC Project [Nob]. The Bloodhound SSC project develops a rocket propelled land vehicle designed to break the land speed record. Detailed are modifications to the automatic stream surface seeding algorithm for large CFD simulation data including: handling large unstructured grids, reducing the memory footprint of the algorithm data structures, and customisation of the distance function used to cluster the simulation data. The modified clustering algorithm is used to capture interesting subsets of the flow and seed stream surfaces in an objective, automatic way. We demonstrate the performance and effectiveness of our framework on the Bloodhound flow simulations and provide domain expert evaluation of the results [ELEC13].

Design of a Flow Visualisation Framework: Chapter 6 discusses research related software. Research software often consists of individual isolated prototype applications. Small proof of concept applications are usually enough for demonstrating new algorithms. The unification of new research algorithms into a cohesive software framework has its advantages. Adding new features to an existing pipeline reduces implementation overhead. The researcher is able to compare and contrast existing or previous work

with new research. Utilising previously implemented techniques, researchers are able to combine visualisation options in new ways that typical research prototypes cannot. The software application is made available to the domain expert for evaluation and future use.

These goals are in part realised by utilising advancements in game design technology, and by leveraging features available with recent graphics hardware. Described is the design of a feature rich flow visualisation software framework, and discussed is the effectiveness and scalability of the approach. It is a system that has been developed and refined for four years [EL13].

Conclusions and Future Work: Chapter 7 presents conclusions and future work. First a summary and a restatement of the main contributions from this thesis are presented. This is followed with a detailed discussion of the conclusions drawn from each chapter focusing on the outstanding challenges and how the following work addresses them. Finally potential future work directions are presented highlighting the challenges which remain for flow visualisation with surfaces.

Mathematical Concepts for Vector Fields: Appendix A describes the mathematical concepts used throughout this thesis. We start with a formal description of the vector field domain, then discuss the structure and interpolation of mesh cells. We then detail the hierarchical spatial hash grid used for fast lookup of unstructured grid cells. Following this we present methods for spline interpolation, and the Runge-Kutta integration techniques. We follow this with a study of the derived fields, finishing with a Table of Notation covering the notation used throughout this thesis.

Gallery of User Options: Appendix B demonstrates the algorithm defined in "Stream Surface Seeding for a Land Speed Record Vehicle" applied with a range of parameters. The motivation for the gallery of results is to demonstrate the variation of the results with change in parameter values. Included are; a gallery of final images of the seeded surfaces, and an equivalent gallery demonstrating the seeding curves used to generate the surfaces.

CHAPTER 2

Surface-based Flow Visualisation

Flow visualisation is a powerful means for exploring, analysing and communicating simulation results. Flow visualisation is characterised by a range of differing techniques such as direct, feature, texture, and geometric-based representations [PVH*03, LHD*04]. Each technique has a range of differing accuracies and speeds [LEG*08]. The phenomena to be studied can be sampled using regular or irregular grids, which can stem from steady state or unsteady flow. There are many challenges to overcome in this field of research. The topic of flow visualisation with surfaces has become an increasingly important field of research in recent years (see Figure 2.1). This is due to the advantages that surface-based techniques offer over more traditional curve-based methods. This provides strong motivation for studying and categorising the breadth and depth of surface-based research for flow visualisation.

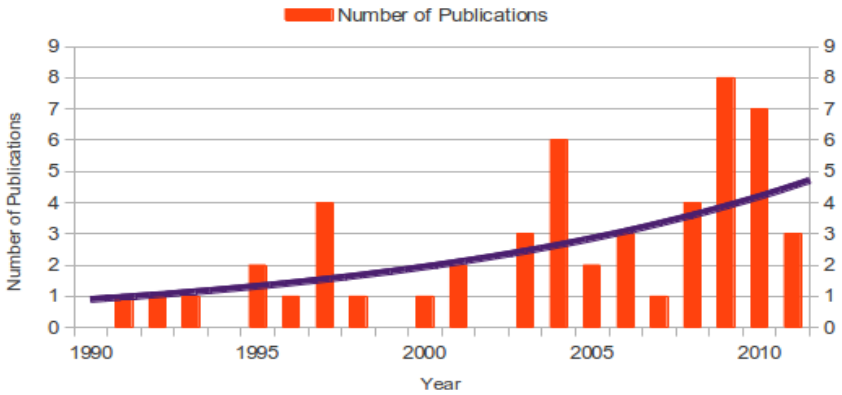


Figure 2.1: *This histogram shows the number of publications per year focused on flow visualisation with surfaces studied in this chapter. It indicates the growing momentum of this topic.*

1.1 Challenges

Surface-based approaches share some common problems associated with flow visualisation in general. Examples of these challenges include: large, time dependent simulation data requiring the utilisation of out of core techniques; and the handling of unstructured data. Surface-based methods also face their own unique challenges which we discuss in more detail throughout the chapter.

Construction Surface construction is a key topic for this survey. Surfaces must represent an accurate approximation of the underlying simulation. Adequate sampling must be maintained while reducing the extra computational overhead associated with over sampling. Resulting meshes must also remain smooth in the presence of various flow phenomena such as vortex cores (the axis about which fluid rotates e.g. the centre of a tornado), and highly divergent or convergent flow (fluid which separates or contracts). A large amount of effort has been put into the creation of various types of surfaces and these form a large portion of this survey. See Section 2 for literature that addresses this challenge.

Occlusion When using surfaces the problem of occlusion occurs frequently. This may stem from multiple surfaces that occlude one another, a large surface that produces self occlusion, or a combination of both. There are several approaches that can be taken depending on the surface type to reduce this problem. A general approach is to use transparency. With integral surfaces, i.e., surfaces to which the flow field is tangent, we have more options. Advanced texture mapping may also be used. Additionally, integral surface seeding positions may be changed to reduce clutter. See Section 3 for literature that addresses this challenge.

Information Content While surfaces offer many advantages in terms of perception, a basic visualisation of the surface alone may not provide sufficient information about the underlying data. For example a stream surface alone does not show the behaviour of inner flow contained within the surface. A review of the research that enhances the resulting visualisation of surfaces is also provided in this survey. See Section 3 for literature that addresses this challenge.

Placement and Seeding Interactive placement is the most common method currently used. There is a strong correlation between seeding and occlusion of integral surfaces. Seeding too many surfaces, or seeding them in such a way that they occupy the same region of the domain leads to high levels of occlusion. The placement of isosurfaces is a function of the selected isovalue. Choosing optimal isovalues is an analogous problem. See Sections 2.4, 2.5, 3.2 and 3.3 for literature which studies these challenges.

Classification			
Constructing Surfaces for Flow Visualisation			
Integral: Stream/Path	Integral: Streak/Time	Implicit	Topological
[Hul92]s, [USM96]s, [SBH*01]s, [GTS*04]s, [STWE07]t, [GKT*08]t, [PCY09]s, [SWS09]s, [MLZ09]t, [PS09]s. [YMM10]t. [SRWS10]s.	[vFWTS08b]t, [KGJ09]t, [BFTW09]t, [MLZ10]t, [FBTW10]t,	[vW93]s, [WJE00]s, [Gel01]s.	[TWHS03]s, [WTHS04]s, [TSW*05]t, [BSDW12]t
Rendering Flow on Surfaces for Visualisation			
Direct	Geometric	Texture: Static Texture	Texture: Dynamic Texture
[PL08]s, [PGL*12]s.	[LMG97]s, [LMGP97]s, [WH06]s, [SLCZ09]s, [BWF*10]s, [HGH*10]t.	[vW91]s, [dLvW95]s, [FC95]t, [MKFI97]s, [BSH97]s, [SK98]t, [Wei07]t, [PZ10]s.	[LJH03]t, [vW03]t, [LvWJH04]t, [LSH04]s, [LWSH04]s, [WE04]t, [LGD*05]s, [LGSH06]s, [BSWE06]s, [LTWH08]t.

Table 2.1: *This table classifies surface techniques into two main categories; Constructing surfaces for flow visualisation and Rendering flow on surfaces for visualisation. The table also sub classifies the surface construction into Integral, Implicit, and Topological, with Integral surfaces further divided between stream/path and streak surfaces. Additional sub classification of this section into point, triangle and quad primitives are shown with colour. The rendering section is sub classified into Direct, Geometric, and Texture techniques, with the texture category further subdivided into static texture and dynamic texture techniques. Additional sub classification of this section into Parameter Space, and Image Space techniques are displayed with colour. An additional suffix is used to represent techniques applied to steady state (s), and time dependent (t) vector fields. Each of the entries are ordered chronologically within each subcategory.*

1.2 Classification

The classification in this chapter represents the subtopics of flow visualisation with surfaces. The classification highlights areas which are more mature and areas which require additional work. Refer to Table 2.1. The two main classifications highlight the differences between applying visualisation techniques to surfaces, and the underlying surface construction. Surface rendering techniques visualise the fluid flow properties on the surface geometry, whereas surface construction techniques represent the fluid flow with the curvature of the surface.

Each of the two classifications are further subclassified. The surface construction is classified into *Integral*, *Implicit*, and *Topological* techniques, while the visualisation of flow on surfaces is divided into *Direct*, *Geometric*, and *Texture*-based techniques. Topological techniques visualise flow topology explicitly using surfaces to do so. Surface construction is also subclassified into triangle, point, and quad-based meshing techniques, while the visualisation of flow on surfaces is subclassified into parameter space-based, and image space-based techniques.

Another possible choice for the visualisation of flow on surfaces is classification into single chart (single parametrisation) and a collection of charts (atlas-based parametrisation). The image space methods would be treated similar to other simple parametrisations such as C-Space (Configuration Space) techniques [MT04]. The subcategories are further divided into steady flow and time dependent flow. The papers within each subcategory are ordered chronologically. We note that this survey does not cover flat or planar surfaces or slices like those described by Laramee [Lar03].

1.3 Contributions and Summary

Surface techniques fall into two main categories: construction of surfaces and visualisation techniques applied to surfaces. Surface construction techniques are a fairly well researched topic. The majority of techniques are variations and extensions of the original Hultquist method [Hul92]. These techniques are either faster, more accurate, cater to large data domains, or address topology. The self occluding problems inherent of surfaces are partially addressed by Löffelmann et al. who effectively create holes in the surface [LMGP97], Theisel et al. [TWHS03] use connectors to represent the separating surface, and the general approach of using transparency to alleviate occlusion e.g. see Sections 2 and 3.

The methods of van Wijk [vW93] and Westermann et al. [WJE00] concentrate on deriving a scalar field from a vector field and then employing isosurface techniques to represent the domain. Although these techniques provide good domain coverage, visual clutter and occlusion can result. The projection of vector information onto a surface by Laramee et al. [LGSH06], improves performance and perception of the flow local to that surface, but the surface occlusion issue remains due to the nature of the image space-based techniques, e.g. see Sections 2.4 and 3.4.

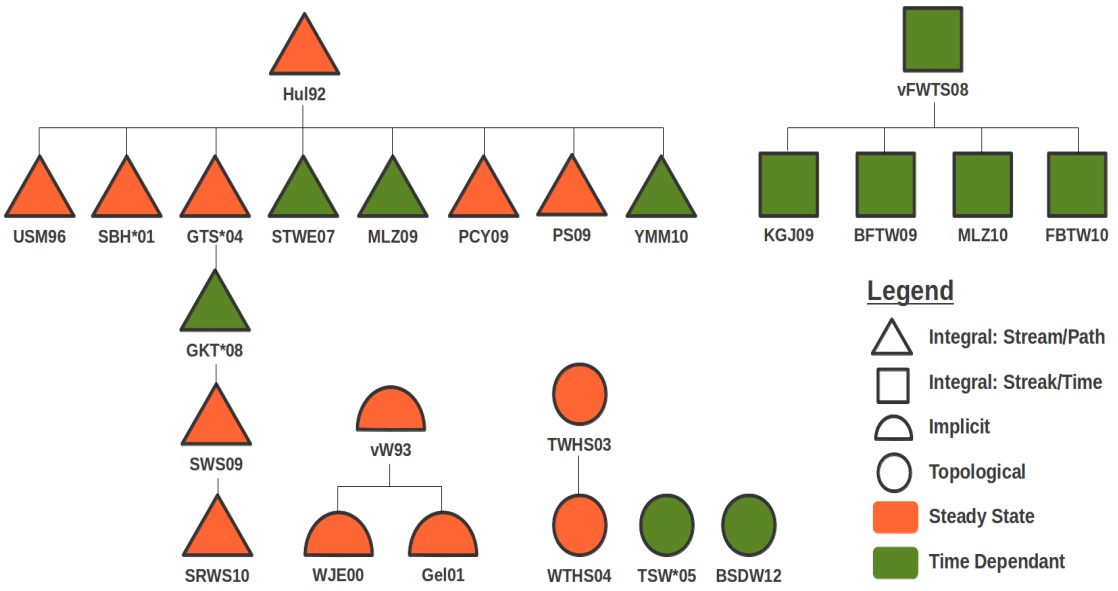


Figure 2.2: *The classifications of construction techniques illustrates a chronological flow of work from author to author. The child parent relationships indicates the key ideas that are progressed. The flow of work diverges and in some cases converges as new concepts are built on top of previous ideas. The charts also show the originating work, and where key work is continued.*

A fairly common theme throughout the different surface types is the application of additional visualisation techniques to enhance the surfaces. Parameter and image space-based techniques are the main focus of these visualisations as they can provide effective interactive solutions. Given a surface (it can be any type, as long as it is manifold) and a vector field defined on it, the flow behaviour can be illustrated with the desired dimension of visual mapping, such as 0D (hedgehogs), 1D (streamlines), or 2D (textures). This enables not only the direct display of the flow data in the Eulerian point of view, or the visualisation of the behaviour of the selected particles in the Lagrangian point of view, but also the complete (dense) image of the flow behaviour over the surfaces. In addition, combined with other conventional visualisation techniques, such as colour coding and animation, more complete flow information including both vector magnitude and orientation, as well as the time varying characteristics, can be conveyed. The main benefits and contributions of this survey survey are:

- A review of the latest research developments in flow visualisation with surfaces.
- A novel classification scheme based on challenges including; construction, rendering, occlusion, and perception. This scheme lends itself to an intuitive grouping of papers that are naturally related to one another.
- The classification highlights both mature areas where many solutions have been provided and unsolved problems.

- A concise overview in the area of flow visualisation with surfaces for those who are interested in the topic and wishing to carry out research in this area.

This report is divided into four main sections: First is a review of the surface construction techniques in Section 2. Then a review of flow visualisation on surfaces in Section 3. An analysis of the different subclassifications is conducted in Section 4 with an emphasis on the initial seed or surface placement/generation, perception, visual clutter and occlusion.

2 Constructing Surfaces For Flow Visualisation

This section discusses the different construction techniques reviewed in this Chapter. Figure 2.2 shows a chronological flow of work from author to author. The child parent relationships indicate the origin and evolution of key ideas. The work diverges as new concepts are built on top of previous ideas.

We start this section with a discussion about flow data and associated challenges, before moving on to the subclassification of primitive types used for the surface mesh representations. Following this we study the range of surface construction techniques. These are divided into *Integral*, *Implicit*, and *Topological* surface construction methods.

2.1 Steady State, Time Dependent and Large Complex Data

Early work focuses on processing steady state simulations which represent a static flow field or single snapshots of flow in time. As the ability to process larger amounts of data increases, the research into processing this data follows. Data sampling becomes denser, the size of the simulation domain increases, and multiple time steps are incorporated and processed. The structure of the data can be complex, incorporating a range of associated scalar quantities representing range of additional attributes.

Velocity data is comprised of a set of x, y, z components for each sample point within the data domain. For example a 3D steady state vector field $\mathbf{v}_s(\mathbf{p}) \in \mathbb{R}^3$ where $\mathbf{v}_s(\mathbf{p}) = \begin{bmatrix}\mathbf{v}_x(x,y,z) & \mathbf{v}_y(x,y,z) & \mathbf{v}_z(x,y,z)\end{bmatrix}$ for $\mathbf{p} \in \Omega$, $\mathbf{v_s} \in \mathbb{R}^3$ and $\Omega \subset \mathbb{R}^3$, where Ω may be a 3D regular, structured, unstructured or irregular grid. For unsteady flow we have a time dependent vector field $\mathbf{v}_t(\mathbf{p}_t) \in \mathbb{R}^3$ where $\mathbf{v}_t(\mathbf{p}_t) = \begin{bmatrix}\mathbf{v}_x(x,y,z,t) & \mathbf{v}_y(x,y,z,t) & \mathbf{v}_z(x,y,z,t)\end{bmatrix}$ for $\mathbf{p}_t \in \Omega_t$, $\mathbf{v}_t \in \mathbb{R}^3$ and $\Omega_t \subset \mathbb{R}^4$.

Our survey discusses work addressing the challenges of both steady and time dependent data. In Table 2.1 we use an 's' suffix to the citation for techniques which process steady state vector fields and a 't' suffix for techniques addressing time dependent data. The trend of moving from steady state toward time dependent data can be observed in the chronological classification of work.

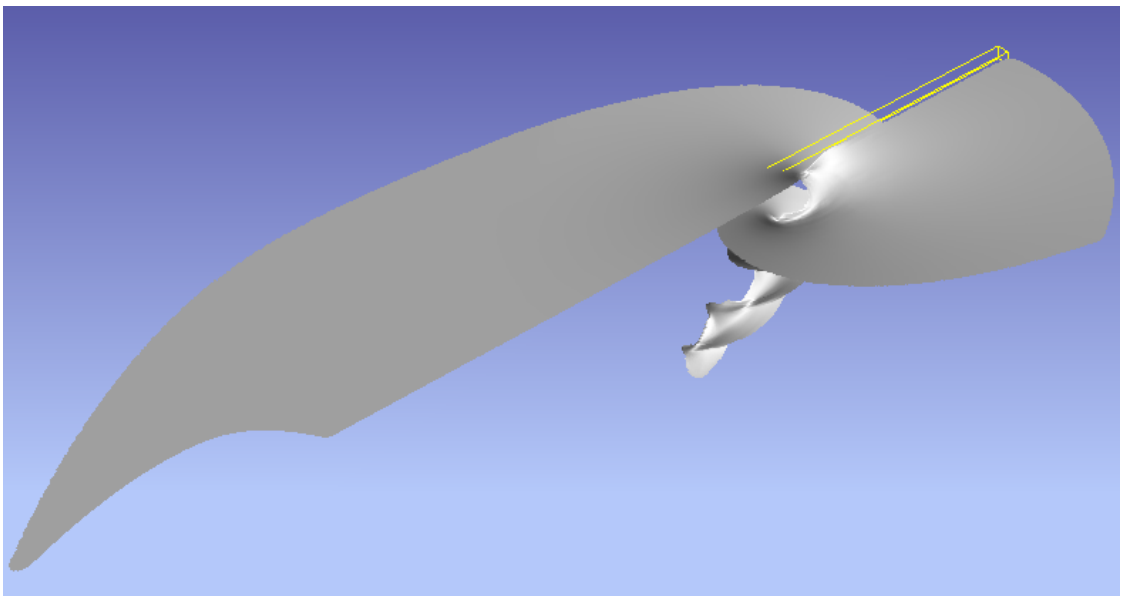


Figure 2.3: *Visualisation of the flow field of a tornado with a point-based stream surface. The stream surface is seeded along a straight line in the centre of the respective image. Image courtesy of D.Weiskopf et al. [STWE07].*

2.2 Point vs. Triangle vs. Quad-based Construction Techniques

OpenGL includes a basic set of primitive constructs exposed by the API. The primitives are used in the construction of meshes which represent the scene to be rendered [Opeb]. Examining the subclassification of surface mesh construction techniques into point, triangle, and quad-based methods yields some interesting insights into these approaches. Point-based surfaces are generally simpler and faster to construct as they do not require any mesh construction computation to represent a closed surface. This approach can be limiting regarding rendering options. Rendering the vertices as simple point sprites, disks, or spheres is effective for dense vertex representations, but gaps or inconsistencies can appear when viewing in close proximity to the surface. Lighting can also be a challenge with this technique as the methods available for normal calculation become limited and increase computation. Another approach to rendering is using image-based techniques which don't explicitly require geometric primitives to render closed surfaces.

Schafhitzel et al. [STWE07] present a point-based stream and path surface algorithm where the vertices for the surface representation are generated on the GPU. With a dense output, each of the vertices and their normals are used to render a closed surface as small, lit, point sprites, as in Figure 2.3. A texture-based closed surface rendering can also be achieved using Line Integral Convolution (LIC) [CL93] performed in image space. With a similar approach, Ferstl et al. [FBTW10] present a streak surface algorithm which has a rendering option using spherical point sprites to represent a closed surface.

To represent a geometric mesh for rendering we must define how the mesh is con-

structed. Of the two common methods for defining a mesh, one is defined by constructing an array of vertices in the correct order for rendering the primitive. This approach can hold redundant instances of the same vertices for a given mesh, with a larger memory footprint and increased data traversing the graphics bus at the cost of valuable bandwidth. In some simple surface construction implementations however this method can be easier to implement.

A second approach to defining a mesh for rendering primitives is the utilisation of an indexing array along with the array representing the vertices of the surface. An additional index array specifies each vertex in the correct order to construct the geometric primitive. In the context of surfaces the result of this approach is a much smaller data array, but the addition of an index array. The index array can be compressed, depending on the quantity of vertices, by using data types requiring less memory such as unsigned characters or unsigned short integers instead of integers. This also significantly reduces the graphics bandwidth.

Another constraint on implementations is the method used for constructing the data and index arrays. The most common mesh primitive used is the triangle. This is the default for surface construction techniques such as isosurfaces as used by van Wijk [vW93] and Westermann et al. [WJE00]. This is a byproduct of early graphics card support for rendering triangles.

The most common approach for meshing integral surfaces with triangles is a greedy minimal tiling strategy as described by Hultquist [Hul92]. Other approaches include; additional processing for streak surfaces where the surface topology changes with time as in the work by Krishnan et al. [KGJ09] as shown in Figure 2.4, and processing of irregular grids such as tetrahedra as described by Scheuermann et al. [SBH*01].

With advancing front integration techniques, a simple approach to generating a mesh is the direct use of the quad patch represented by the bounding streamlines and timelines. This approach initially requires less computational expense, however dealing with issues of sheering quads, t junctions and additional normal computations (one per quad corner rather than one per triangle) can have a significant impact. Refer to Figure 2.5 and see McLoughlin et al. [MLZ09, MLZ10] and Schneider et al. [SWS09]. Peikert and Sadlo [PS09] construct their surface geometry in an incremental manner advancing from the initial curve structure attempting to avoiding the issue of quad sheering. The algorithm by van Gelder et al. [Gel01] also lends itself to qua-based meshing due to the connectivity of the curvilinear grids.

2.3 Integral Construction Techniques

In this subsection we present an overview of integral surface construction techniques. This work is subdivided into stream/path, and streak/time surface construction algorithms. The similarity between the stream and path surface construction provides a natural classification for discussion in the next subsection. Following the review of stream/path surfaces we then present a study of streak surface construction. The main

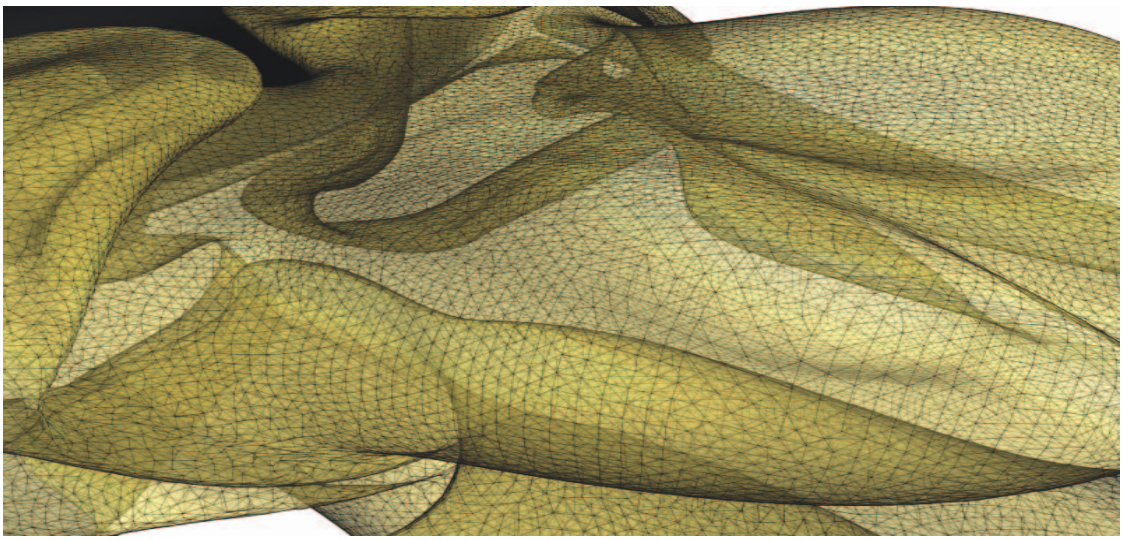


Figure 2.4: *Time surface mesh in the Ellipsoid dataset. Although the surface has undergone strong deformation, the mesh remains in good condition. Image courtesy of C.Garth et al. [KGJ09].*

challenges addressed by the literature in this subsection are accurate surface construction, performance time, and continuous representations.

Stream/Path Surface Construction

A **streamline** is a curve which is tangent to the velocity field at every point along its length. A Streamline is the trace of a massless particle from an initial location (seed point). We define a massless particle as being infinitely small, having no kinetic energy, but still subject to all frictional forces of the fluid, i.e., the particle exclusively follows the fluid elements which it neighbours. Streamlines show the direction fluid flow within a steady state flow domain. If $\mathbf{v}(\mathbf{p})$ is a three dimensional vector field, the streamline through a point, $\mathbf{p}_0$, is the solution $I_s(\mathbf{p}_0,t)$ to the differential equation:

$$\frac{d}{dt}I_s(\mathbf{p}_0,t) = \mathbf{v}(I_s(\mathbf{p}_0,t)) \tag{2.1}$$

where, in the case of a steady state flow field, t is time. The initial condition is $I(\mathbf{p}_0,0) = \mathbf{p}_0$. The case of a static flow field is treated the same [GKT*08]. A **stream surface** is the trace of a one dimensional seeding curve, C, through the flow. The resulting surface is everywhere tangent to the local flow. A stream surface, S, is defined by:

$$S(s,t) := I_s(C(s),t) \tag{2.2}$$

S is the union or continuum of integral curves passing through the seeding curve C. $S(s,-)$ coincides with an individual integral curve, and $S(-,t)$ coincides with individual time lines [GKT*08].

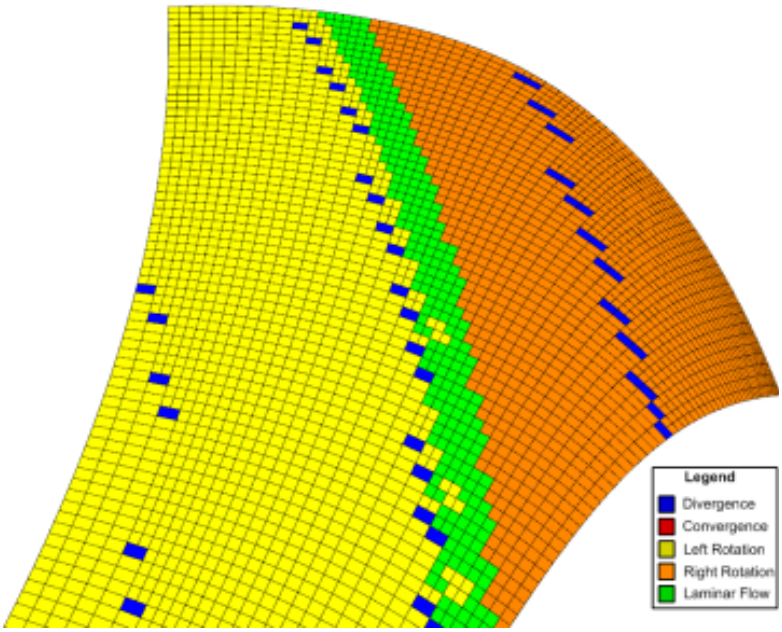


Figure 2.5: *The algorithm by McLoughlin et al. [MLZ09] handles widely diverging flow while maintaining the desired organised advancing front. This surface is coloured according to the underlying flow characteristics: left rotation is mapped to yellow, right rotation is mapped to orange, parallel flow is mapped to green, divergence is mapped to blue, convergence is mapped to red. Image courtesy of R.S.Laramee et al. [MLZ09].*

Since there is no normal component of the velocity along streamlines and stream surfaces, mass cannot cross their boundary and therefore they are useful for separating distinct regions of similar flow behaviour. In practical applications, a discretised approximation of the stream surface is constructed by tracing discretised seeding curves through the vector field using integration methods such as the fourth-order Runge-Kutta integration scheme.

Hultquist introduces one of the first methods [Hul92] for the generation of stream surface approximations. This technique includes strategies for controlling the density of particles across the advancing front. Points can be added to the advancing front when the sampling rate becomes too sparse and removed when it becomes too dense. Neighbouring pairs of streamlines are tiled with triangles to form ribbons. These connected ribbons then form the surface. Ribbons which encounter rapid divergence of flow may be torn/ripped to allow the surface to flow around an obstacle. The separate portions of the surface are then computed independently. Hultquist builds on work by Belie [Bel87] and Kerlick [Ker90] who describe narrow stream ribbon methods, and Schroeder et al. [SVL91] who describe a stream primitive called the stream polygon.

Ueng et al. [USM96] expand the stream surface work to stream ribbons, stream tubes, and streamlines on unstructured grids. The authors build on the stream polygons

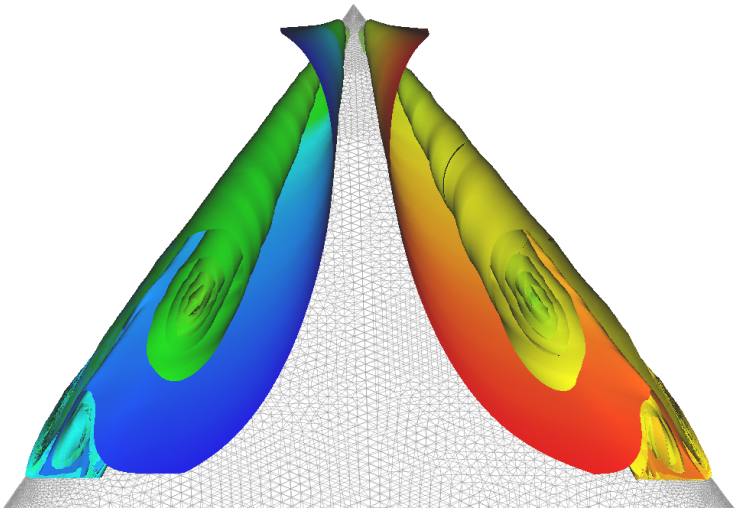


Figure 2.6: *The formation of vortices at the apex of a delta wing illustrated with the use of a stream surface. Image courtesy of C.Garth et al. [GTS*04].*

algorithm by Darmofal and Haimes [DH92], the research by Ma and Smith [MS93], and the steam ribbons of Pagendarm [PW94]. This paper describes extending the techniques to unstructured grids by converting the physical coordinate system to a canonical coordinate system. The main idea is the use of a specialised fourth-order Runge-Kutta integrator which requires only one matrix vector multiplication and one vector vector addition to calculate the successive streamline vertices. This technique significantly simplifies the construction of the geometric primitives, reducing the computational cost and therefore improving speed.

Scheuermann et al. [SBH*01] present a method of stream surface construction on tetrahedral grids. The technique propagates the surface through the tetrahedral grid, one tetrahedron at a time, calculating on the fly where the surface intersects the tetrahedron. This approach enables the inclusion of topological information from the cells such as singularities. When the surface passes through the tetrahedron, the curve segments end points are traced as streamlines through the next cell. For each point on a streamline, a line is added connecting it to its counterpoint. These are then clipped against the faces of the tetrahedron cell and the result forms the boundary of a polygonal surface within the cell. This method is inherently compatible with multi resolution grids and handles increased grid resolution in intricate flow regions. This work is a significant improvement over the previous irregular grid work, improving surface construction within complicated areas, typically of more interest in fluid dynamics.

Garth et al. [GTS*04] improve on the Hultquist method and showed how to obtain surfaces with higher accuracy in areas of intricate flow. See Figure 2.6. The improvements are achieved by employing arc length particle propagation and additional curvature-based front refinement. They also considered visualisation options such as

colour mapping of vector field related variables going beyond straightforward surface rendering. A novel method to determine boundary surfaces of vortex cores and a scheme for phenomenological extraction of vortex core lines using stream surfaces is discussed and its accuracy is compared to one of the most established standard techniques.

Next we review the concept of a path surface. We start by describing a pathline. A **pathline** or particle trace is the trajectory that a massless particle takes in time dependent fluid flow. If $\mathbf{v}(\mathbf{p},t)$ is a three dimensional vector field for $\mathbf{p}$ in domain $\Omega \in \mathbb{R}^3$ and t in a time interval $[T_0,T_1]$ the pathline $I_p(\mathbf{p}_0,t_0;t)$ passing through $\mathbf{p}_0$ at time t_0 is the solution to the ordinary differential equation:

$$\frac{d}{dt}I_p(\mathbf{p}_0,t_0;t) = \mathbf{v}(I_p(\mathbf{p}_0,t_0;t),t) \tag{2.3}$$

with the initial condition $I_p(\mathbf{p}_0,t_0;t_0) = \mathbf{p}_0$. A **path surface** is the trajectory of a massless curve, C, in time dependent fluid flow. A path surface, P, is defined by:

$$P(s,t) := I_p(C(s),t_0;t) \tag{2.4}$$

Where P is the union or continuum of pathlines passing through the seeding curve, C, at time t_0. $S(s,-)$ coincides with an individual integral curve, and $S(-,t)$ coincides with an individual timeline [GKT*08]. The first example of this is the work by Schafhitzel et al. [STWE07] who introduce a point-based algorithm for stream and path surface construction and rendering.

Schafhitzel et al. combine and build on three specific areas: stream surface computation Hultquist [Hul92], rendering of point-based surfaces Zwicker et al. [ZPKG02], and texture-based flow visualisation on surfaces (Weiskopf et al. [WE04]). The stream/-path surface generation is modified to run on the GPU in a highly parallel fashion. Seed points are generated and integrated through the vector field. To maintain a roughly even density, integrated points along the advancing front are inserted and removed. The surface rendering method is based on Point Set Surfaces (PSS) and is extended to include stored connectivity information, this enables quick access to neighbouring points. The authors' approach to the hybrid object/image space LIC method displays clear line patterns which show a choice of path lines.

More recently, Garth et al. [GKT*08] replaced the advancing front paradigm by an incremental time line approximation scheme. See Figure 2.7. This allows them to keep particle integration localised in time. The authors propose a decoupling of the surface geometry and graphical representation, and a curve refinement scheme which is used to approximate time lines, yielding accurate path surfaces in large time dependent vector fields.

Following recent work by Bachthaler and Weiskopf [BW08] who describe the use of tracing structures perpendicular to the vector field to generate animated LIC patterns orthogonal to the flow direction, and work by Rosanwo et al. [RPH*09] which introduces dual streamline seeding based on streamlines orthogonal to the vector field, Palmerius

et al. [PCY09] introduce the concept of perpendicular surfaces. These surfaces are perpendicular to the underlying vector field.

The authors describe the common properties of such surfaces and discuss the issues of non zero helicity density, and stop conditions. The construction method starts at a predefined seed point propagating outwards in a clockwise spiral fashion where each new point is integrated perpendicular to the flow field for a given distance. The stop criteria are: length limit, accumulated winding angle limit, and maximum orientation error as a result of vector fields with non zero helicity. Convergence or divergence is characterised by cone shaped surfaces. A combination of both results are saddle shaped surfaces. Vortices distort the surfaces by tearing them apart and producing a fan like pattern. A fast approach for generating the surfaces and stop conditions is also described.

Extending the previous work by Garth et al [GKT*08], Schneider et al. [SWS09] produce a more accurate and smoother timeline interpolation using a fourth-order Hermite interpolation scheme when adjusting the advancing front density. The Hermite interpolation between streamlines requires the covariant derivatives to be calculated with respect to s (streamline) and t (timeline). An additional surface accuracy error criterion is used to dictate when coarsening or refinement takes place. The error-based refinement strategy splits a ribbon when the local interpolation error exceeds a given bound. This error is estimated directly by seeding a new short streamline from a position between neighbouring streamlines at time $t_n - 1$ integrating it to time t_n.

Alternatively, McLoughlin et al. [MLZ09] describe a simple and fast CPU-based method for creating accurate stream and path surfaces using quad primitives. The au-

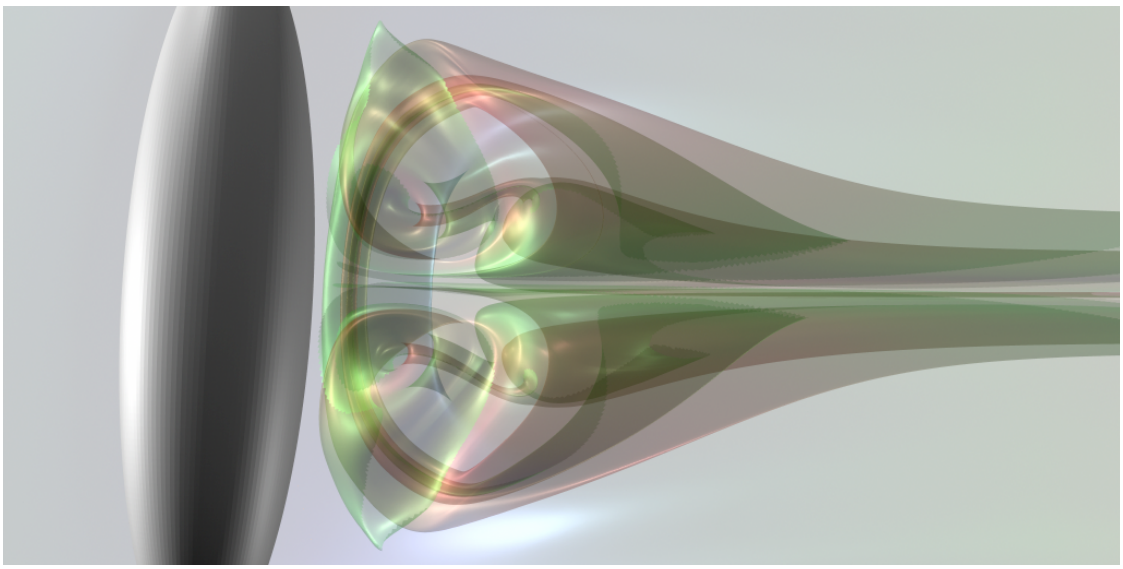


Figure 2.7: *Path surface visualisation of vortex shedding from an ellipsoid. The transparent surface consists of 508,169 triangles. Different layers identified by colour mapping. Image courtesy of C.Garth et al. [GKT*08]. © IEEE/TVCG.*

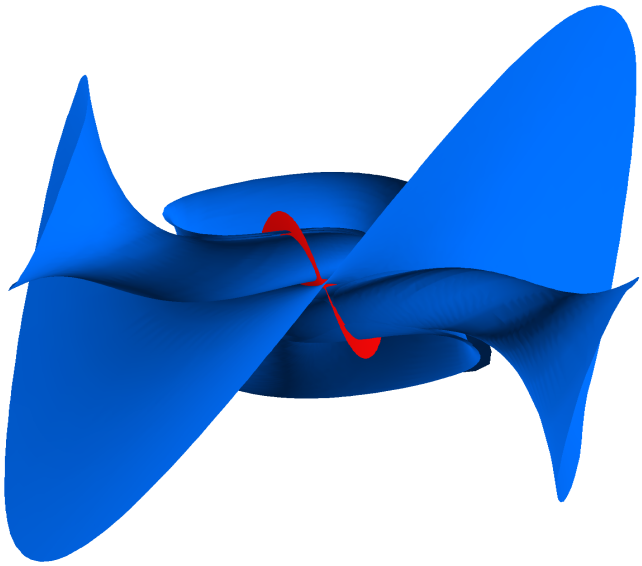


Figure 2.8: *2D manifolds visualising the critical point of the Lorenz dynamical system. Image courtesy of R.Peikert et al. [PS09].*

thors propose a method which is based on a small set of simple, local operations performed on quad primitives, requiring no global remeshing strategy. To handle divergent, convergent, and rotational flow, the sampling rate of the advancing front is updated. The quad is either divided (in areas of divergence) or collapsed (in areas of convergence). For curvature, a test of the advancing front rotation is conducted. If true then the advancing front integration step size is reduced by a factor dependent on the amount of rotation.

Using topology for the construction and placement of flow geometry, Peikert and Sadlo present topology relevant methods for constructing seeding curves to produce topologically-based stream surfaces [PS09]. The authors build on work by Garth et al. [GTS*04] expanding the notion of feature visualisation applying stream surfaces to a range of singularities and periodic orbits. The discretised offset curves constructed at the topological structures are used to initialise the stream surface propagation. See Figure 2.8. The authors construct their stream surface from quads which are divided in areas of divergence to maintain a consistent mesh topology. This is achieved by subdividing between neighbouring nodes with a cubic interpolant after tracing back a fixed number of steps. The nodes of the mesh retain a number of attributes representing the flow field, which are used for controlling the growth of the surface and texturing.

The work by Yan et al. [YMM10] proposes a number of surface surgery operations during integration to reveal the fractal geometry (thin sheet rotating around and tending to the attractors) of the strange attractors in 3D vector fields which previously were difficult to visualise. Their method consists of three major steps. First a polygonal surface is advected and deformed according to the vector field. This polygonal surface is initialised as some regular shape, such as a torus, which neglects the fractal dimension of the strange attractor. Second, due to the possible high distortion, the polygonal surface

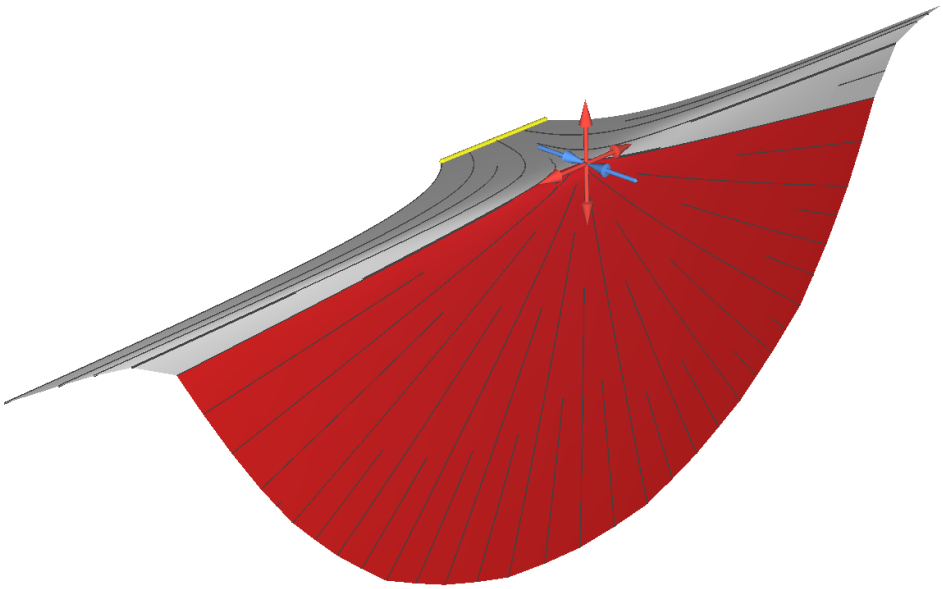


Figure 2.9: *A stream surface in a linear vector field with highly diverging streamlines where the angle criterion (130 degrees) for splitting the surface fails because the surface does not run into the saddle. The red part of the surface would have been left out if the angle criterion were to be used. Image courtesy of D.Schneider et al. [SRWS10].*

may need refinement. In this step, a GPU-based adaptive subdivision (i.e. edge division) of mesh is applied to preserve the necessary features. A mesh decimation on the CPU may also be conducted to reduce the resolution of uninteresting portion of the surface for memory efficiency. Third, a mesh retiling is performed to maintain the consistent triangulation of the thin sheet structure when approaching the attractor and to correct the self intersection artefacts. This method has shown its utility through examples with strange attractors and is expected to apply to other integral surface computations.

Another extension to steam surface algorithms inspired by Peikert and Sadlo [PS09], is the work by Schneider et al. [SRWS10] whose algorithm detects singularities within the flow field and deals with them appropriately, rather than the current methods of continuous refinement or splitting the surface. The authors use a preprocessing step to generate the required topological information. The stream surface algorithm then detects intersections with the separating two dimensional manifold of a saddle point. The resulting surface will either follow a new direction appropriate to the local vector field when encountering a node saddle (See Figure 2.9) or split when encountering a spiral saddle.

Streak/Time Surface Construction

The main challenges addressed by the methods presented here are computational time and maintaining a continuous dynamic surface.

A **streakline** is the line joining a set of massless particles that have all been seeded successively over time at the same spatial location in time dependent flow. Dye steadily injected into the fluid at a fixed point extends along a streakline. If $\mathbf{v}$ is a three dimensional vector field defined over a domain $\Omega \in \mathbb{R}^3$ and time interval $[T_0, T_1]$ to find the streakline $L(\mathbf{p}, T_0, T_1; s)$ for particles seeded at $\mathbf{p}_0$, starting at time T_0, as it appears at time T_1, we must solve separately the pathline equation for $I_p(\mathbf{p}_0, t_0; t)$ for each $t_0 \in [T_0, T_1]$, and then let $L(\mathbf{p}, T_0, T_1; s) = I_p(\mathbf{p}_0, T_1 - s; s)$, for $s \in [0, T_1 - T_0]$. If seeded at the same location in a steady state flow field streamlines, pathlines and streaklines are identical.

A **streak surface** is the smooth union of streaklines from seeding locations along a continuous curve, C. A streak surface, K, is the union of all particles emanating continuously from a parametrised curve, $C(u)$, over time interval $[t_0, t_1]$ and moving with the flow from the time of seeding t. In terms of individual streaklines it can be described as:

$$K(u, T_0, T_1; t) := L(C(u), T_0, T_1; t) \tag{2.5}$$

Introducing the first streak surface approximation, Von Funck et al. [vFWTS08b] represent smoke structures as a triangular mesh of fixed topology, connectivity and resolution. The transparency of each triangle is represented by α where:

$$\alpha = \alpha_{density}\ \alpha_{shape}\ \alpha_{curvature}\ \alpha_{fade} \tag{2.6}$$

The density component $\alpha_{density}$ is a representation of the smoke optical model by considering the triangle primitive to be a small prism filled with smoke. The shape component α_{shape} of the triangle is defined as the ratio of its shortest edge to the radius of its circumcircle and represents its distortion. The curvature $\alpha_{curvature}$ is represented by the local mean surface curvature. The fade α_{fade} is defined as an increase in transparency over time. To effectively render the transparency at real time frame rates a depth peeling algorithm is utilised. The authors demonstrate modifications to the algorithm to mimic smoke nozzles and wool tufts. This technique is the first step in generating streak surfaces, and addresses the occlusion issues associated with complex flow structures represented by surfaces.

Focusing on performance of large, time varying vector fields, Krishnan et al. [KGJ09] propose a method for time and streak surface generation. Their approach enables parallelisation by decoupling the surface advection and surface refinement. The authors build on work by Von Funck [vFWTS08b] with extensions to time surfaces while parallelising the pipeline and improving the meshing scheme using techniques described by Bridson [Bri03]. This paper describes the algorithm with respect to time surfaces and then explores the extension to streak surfaces. A time surface is generated in two steps. First each point of the initial surface mesh is advected over the next time interval. The mesh is then passed to the adaptation phase where three basic operations, edge split,

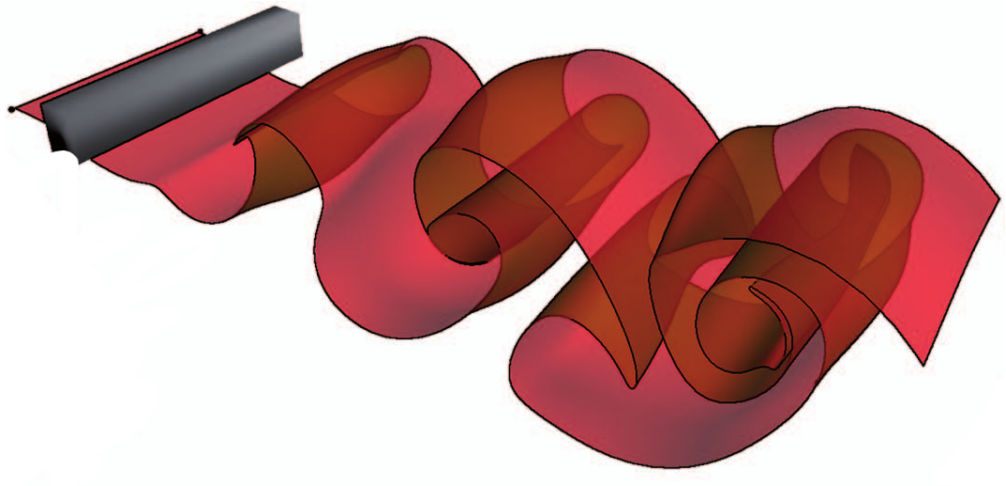


Figure 2.10: *The visualisation of a transparent streak surface rendered using depth peeling and generated on the GPU. Image courtesy of H.Theisel et al. [BFTW09]. © IEEE/TVCG.*

edge flip, and edge collapse are applied to refine the surface. To prevent irreparable changes to the mesh, the integration time step is automatically chosen requiring that no vertex moves further from its current position than a predetermined function of maximum velocity. Streak surface evolution is refined using a similar approach accounting for the new particles seeded continuously from the seeding curve. The surface visualisation uses a combination of texture mapping, lighting effects, and depth peeling for transparency. The use of these effects helps with occlusion and depth perception.

Continuing in the same theme, Burger et al. [BFTW09] describe two methods of streak surface construction for the visualisation of unsteady flow. Building on work by Von Funck et al. [vFWTS08b] the authors present the first real time approach for adaptive streak surface integration and high quality rendering. See Figure 2.10. The first approach computes a quad-based surface where each quad patch is independent of all others. This independence enables parallel processing and rendering of each patch on the GPU. The refinement of these patches is performed independently and is based on an area criterion. If the criterion threshold is met the quad patch is split along the longest edge and its opposite edge, forming two new independent patches. The patches are rendered directly from the vertex buffer using a two pass approach to fill gaps left by the refinement process.

The second approach computes a point-based interconnecting triangular mesh which is modified during the refinement process. Each advected timeline is stored in its own vertex buffer in order, and refined in every time step. The refinement process is completed in three passes: time line refinement, connectivity update, streak line refinement. When inserting points the location is determined by fitting a cubic polynomial and bisecting it equally between the two diverging points. The connectivity of the mesh is

Figure 2.11: *Particle-based surface visualisation. Red particles correspond to points on the separating surface. Green particles serve as context information. They correspond to points on time surfaces, which are released from the planar probe at a fixed frequency. Image courtesy of H.Theisel et al. [FBTW10]. © IEEE/TVCG.*

then updated by searching the previous and next timelines for any given point's nearest neighbour. The third pass computes the maximum Euclidean distance between neighbouring timelines. The complete timeline is then added or removed. The mesh is then rendered after a final pass computes the mesh triangulation.

Extending their work on quad-based stream and path surfaces, McLoughlin et al. [MLZ10] present a novel streak surface algorithm using quad primitives. The refinement of the surface is achieved by performing local operations on a quad by quad basis. Quads may be split or merged to maintain sufficient sampling in regions of divergence and convergence. Shear flow is handled by updating the topology of the mesh to maintain fairly regular quads. This method is designed for and implemented on the CPU and generally achieves interactive frame rates.

Following the collection of works which define methods for streak surface construction, Ferstl et al. [FBTW10] introduce real time construction and rendering of surfaces, which represent Lagrangian Coherent Structures (LCS), in conjunction with the rendering of the streak surface particles. See Figure 2.11. This technique interactively displays the separation surfaces leading to new possibilities of studying complex flow phenomena. The user can interactively change the seeding parameters, and visually display the separation surfaces, resulting in a visually guided exploration of separation surfaces in 3D time dependent vector fields.

LCS are computed by extracting the ridges in the finite time Lyapunov exponent (FTLE) field. The paper builds on the work by Sadlo and Peikert [SP07] who describe a filtered ridge extraction technique based on adaptive mesh refinement. The method enables a substantial speed up by avoiding the seeding of trajectories in regions where no ridges are present or do not satisfy the prescribed filter criteria such as a minimum

finite Lyapunov exponent.

The finite time Lyapunov exponent (FTLE) [Hal01] quantifies the local of separation behaviour of the flow. It is used to measure the rate of separation of infinitesimally close flow trajectories. The pathline solution $\mathbf{p} = I_p(\mathbf{p}_0, t_0; t)$ of Section 2.3 for fixed times t_0 and t can be considered as a flow map from a position $\mathbf{p}_0$ to the pathline position $\mathbf{p}$ where it is advected by the flow at time t. Using the flow map, the *Cauchy Green deformation tensor field*, $\mathbf{C}_{t_0}^t$ is obtained by left multiplying the Jacobian matrix of the flow map with its transpose [Mas99]:

$$\mathbf{C}_{t_0}^t(\mathbf{p}) = \left[\frac{\partial(\mathbf{p}_0, t_0; t)}{\partial(\mathbf{p}_0)}\right]^T \left[\frac{\partial(\mathbf{p}_0, t_0; t)}{\partial(\mathbf{p}_0)}\right] \tag{2.7}$$

From this, the FTLE is computed by:

$$\mathbf{FTLE}_{t_0}^t(\mathbf{p}) = \frac{1}{t - t_0} \ln \sqrt{\lambda_{max}(\mathbf{C}_{t_0}^t(\mathbf{p}))} \tag{2.8}$$

where $\lambda_{max}(\mathbf{M})$ is the maximum eigenvalue of matrix $\mathbf{M}$ [Hal01].

FTLE requires the choice of a temporal window, the effect of a change in the time window length has not been studied sufficiently [PPF*10]. A common use of FTLE is to extract *Lagrangian Coherent Structures* (LCS). LCS are extracted from an FTLE field by ridge extraction [SP07].

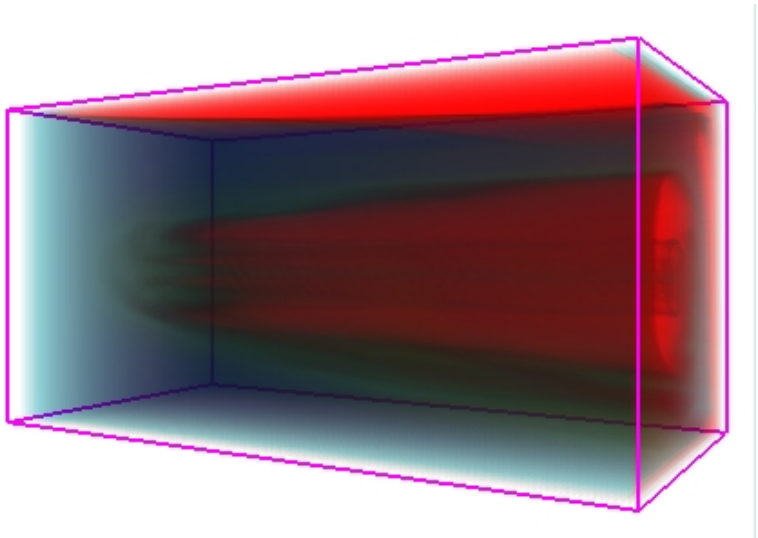


Figure 2.12: *Dense flow fields are first converted into a scalar field, and then displayed and analysed by means of level sets in this field. Image courtesy of R.Westermann et al. [WJE00].*

2.4 Implicit Construction Techniques

This section reviews a set of surface construction techniques which are described by solving some function of the underlying flow field. The motivation for this type of technique include avoiding compound error associated with integration schemes and meshing challenges resulting from convergent, divergent and shear flow. The work in this area is limited to stream surface representations in steady state velocity data.

van Wijk [vW93] introduces a method for the global representation of the stream surfaces as implicit surfaces $f(\mathbf{p}) = C$. Once f is defined at the boundary, the definition is then extended to the interior of the domain by specifying that it is constant on streamlines. C can be varied to efficiently generate a family of stream surfaces. The originating curves are defined at the boundary by the value of f. This method greatly differs from the advancing front methods introduced by Hultquist [Hul92]. Two methods are presented to derive f; The first is based on solving the convection equations, and the second is based on backward tracing trajectories from grid points. The 3D stream function defines a scalar field from which traditional isosurface extraction techniques are then used to create the stream surfaces.

Taking this concept a step further, Westermann et al. [WJE00] present a technique for converting a vector field to a scalar level set representation. See Figure 2.12. An analysis of the subsequent distorted level set representation of time surfaces is conducted before combining geometrical and topological considerations to derive a multi scale representation. This is implemented with the automatic placement of a sparse set of graphical primitives, depicting homogeneous streams within the fields. The final step is to visualise the scalar field with iso surfaces. The advantage of this technique is full domain coverage as the van Wijk [vW93] method constructs surfaces only where intersections with the boundaries occur.

With a different approach van Gelder. [Gel01] introduce a semi global method which does not suffer from the compound error from integral surface generation or computational overhead and error seen in the global approaches. Stream surfaces are constructed on 3D curvilinear grids which satisfy the constraints of a region expressed as integrals, instead of solving a local ordinary differential equation. The constraints are expressed as a series of solvable quadratic minimisation problems. The solution exploits the fact that the matrix of each quadratic form is tridiagonal and symmetric. The author describes the transformation of the curvilinear grid into parameter space to simplify the stream surface construction problem.

2.5 Topological Surface Construction Techniques

The challenge of topology-based methods is to separate or segment the flow into areas of similar behaviour. As part of this process singularities and separatrices are extracted from the flow field. In steady state flow the separatrices are stream surfaces. The topological structures can also useful for supporting other flow visualisation methods, and is

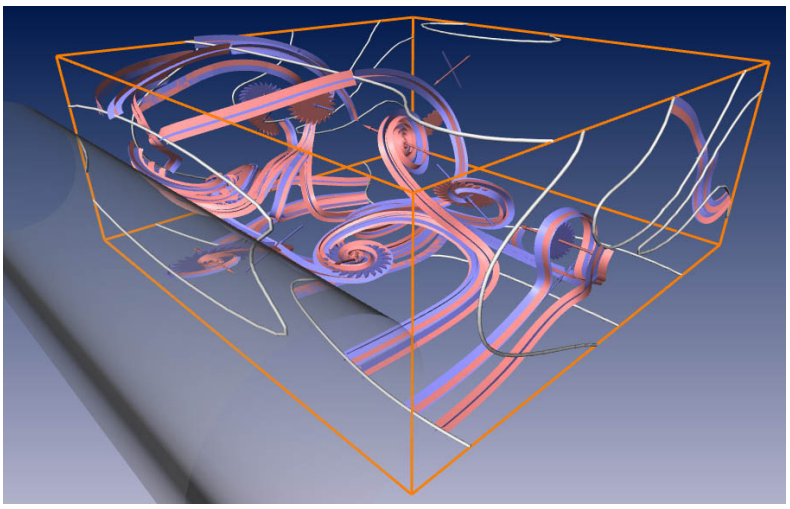


Figure 2.13: *Topological skeleton showing saddle connectors, singularities and boundary switch connectors. Image courtesy of H.Theisel et al. [WTHS04].*

the inspiration for techniques is this section.

Theisel et al. [TWHS03] present an approach for constructing saddle connectors in place of separating stream surfaces is a significant effort to help alleviate occlusion. A **saddle connector** is a streamline which connects two saddle points. Building on work by Theisel and Seidel [TS03] the authors apply saddle connectors in three dimensions. This work is extended by Weinkauf et al. [WTHS04] who introduce the concept of separating surfaces originating from boundary switch curves. A **boundary switch curve** is a curve generated at the domain boundary where inflow changes to outflow or vice versa e.g., flow is parallel with the boundary surface. This is achieved by joining saddle points to boundary switch curves, or between each other, using a type of streamline called boundary switch connectors. The idea of using streamline connectors in place of separating surfaces reduces visual clutter as can be seen in Figure 2.13.

Inspired by Theisel and Seidel's work on tracking features in 2D [TS03], Theisel et al. [TSW*05] introduce a method for visualising the propagation of vortex core lines over time. The contextual surfaces are shown emanating from the vortex core lines in Figure 2.14. Two 4D vector fields are computed which act as feature flow fields such that their integration surfaces (e.g. stream surfaces) provide the vortex core structures. The feature flow field is equivalent to the parallel vector (PV) approaches by Peikert and Roth [PR99]. In addition, this work describes a method to extract and classify local bifurcations of vortex core lines in space time through the tracking and analysis of PV lines in the feature flow field.

Avoiding the integration of hyperbolic trajectories by replacing them with intersections of LCS while utilising LIC to reveal the tangential dynamics, Bachthaler et al. [BSDW12] stack 2D vector fields according to time to generate a 3D space time vector field. The LCS ridge structures are computed from an FTLE scalar field generated in

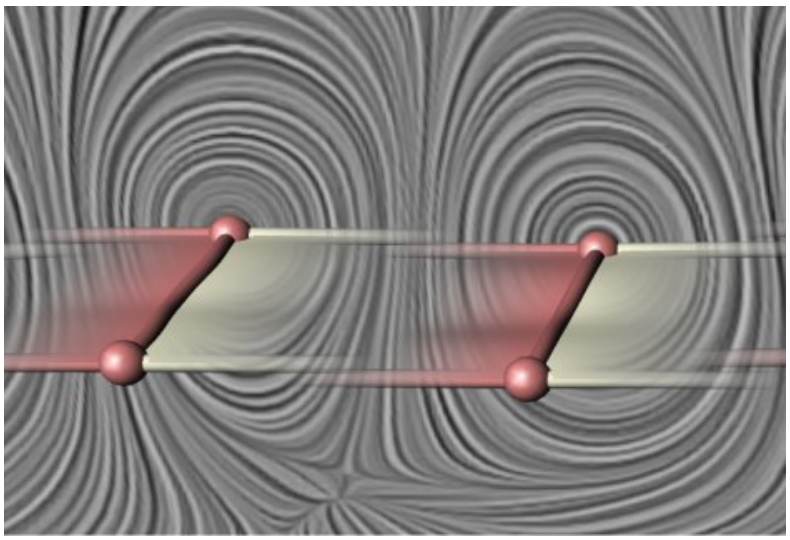


Figure 2.14: *Time surfaces shown as contextual information emanating from the visualised vortex core lines. Image courtesy of H.Theisel et al. [TSW*05].*

three dimensions. The hyperbolic trajectories are mapped to saturation. Visualising the LCS dynamics the authors apply the method described by Weiskopf et al. [WE04]. To address the problem of occlusion in the space time visualisation of the LCS, the paper describes restricting the visualisation to bands around the LCS intersection curves. The authors adopt the concept of hyperbolic trajectories and space time streak manifolds.

3 Rendering Flow on Surfaces for Visualisation

This section presents a survey of techniques that enhance the rendering and visualisation of surfaces used for flow visualisation. Figure 2.15 show a chronological flow of techniques demonstrating the child parent relationships and key ideas that are progressed. The charts also show the originating work, and where key work is continued.

We start this section with a discussion about the conceptual differences of *Parameter Space* and **Image Space** techniques. We then examine the rendering techniques. The techniques in this section are classified into **Direct**, **Geometric** and **Texture**-based. The texture-based subsection is further divided into static and dynamic type textures.

3.1 Parameter Space and Image Space Techniques

One approach to applying texture properties on surfaces is via the use of a parametrisation. Applying textures to surfaces becomes particularly suitable when the whole surface can be parametrised globally in two dimensions as shown by Forssell and Cohen [FC95]. The drawbacks with this approach include challenges such as distorted textures as a re-

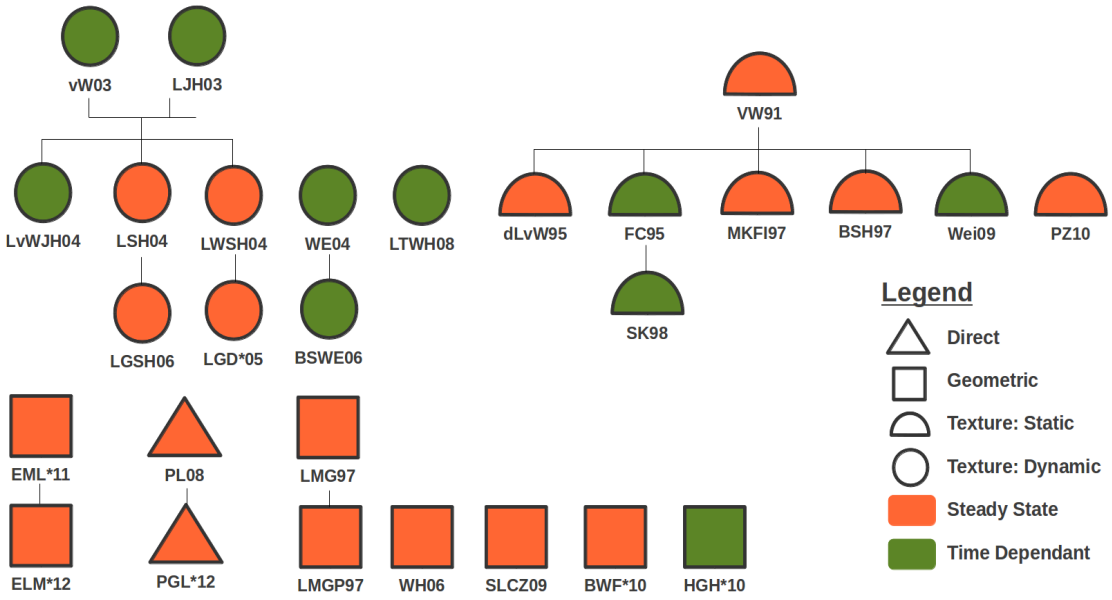


Figure 2.15: *This classification of rendering techniques show a chronological flow of work from author to author. The child parent relationships indicates the key ideas that are progressed. The flow of work diverges and in some cases converge as new concepts are built on top of previous ideas. The charts also show the originating work, and where key work is continued.*

sult of the mapping between object space and parameter space. A global parametrisation for many types of surface is not available such as isosurfaces generated from marching cubes algorithms.

A more recent approach is the use of image space techniques to accelerate computation. The general approach is to project the surface geometry to image space and then apply a series of image space techniques. These techniques can range from advecting dense noise textures [LJH03] to rendering attributes from the underlying data to the surface such as streamlines [SLCZ09], or illustrating various perceptual attributes of the surface such as silhouette edge highlighting [HGH*10].

3.2 Direct Rendering Techniques

Direct visualisation techniques are the most primitive methods of flow visualisation. Typical examples involve placing an arrow glyph at each sample point in the domain to represent the vector data or mapping to some scalar attribute of the local vector field. Direct techniques are simpler to implement and enable direct investigation of the flow field. However, these techniques may suffer from visual complexity and imagery that lacks in visual coherency. They can also suffer from serious occlusion problems when applied to 3D datasets. This idea provides motivation for the work classified in this section.

Extending the direct visualisation paradigm, combining it with clustering techniques and utilising image space methods, Peng and Laramee introduce a glyph placement technique performed in image space which visualises boundary flow [PL08]. This concept builds on work by Laramee et al. [LvWJH04] by projecting the visible vector field from object space to image space, then generating evenly spaced glyphs on a regular grid. The glyphs are a clustered approximation of the underlying vector field mesh resolution. Calculated on the fly, this algorithm is efficient and can handle large unstructured, adaptive resolution meshes.

Following their previous image space work [PL08], Peng et al. [PGL*12] present a novel, robust, automatic vector field clustering algorithm that produces intuitive images of vector fields on large, unstructured, adaptive resolution boundary meshes from CFD. Their bottom up, hierarchical approach is the first to combine the properties of the underlying vector field and mesh into a unified error driven representation. See Figure 2.16. Clusters are generated automatically, so no surface parametrisation is required, and large meshes are processed efficiently. Users can interactively control the level of detail by adjusting a range of clustering distance measures. This work also introduces novel visualisations of clusters inspired by statistical methods.

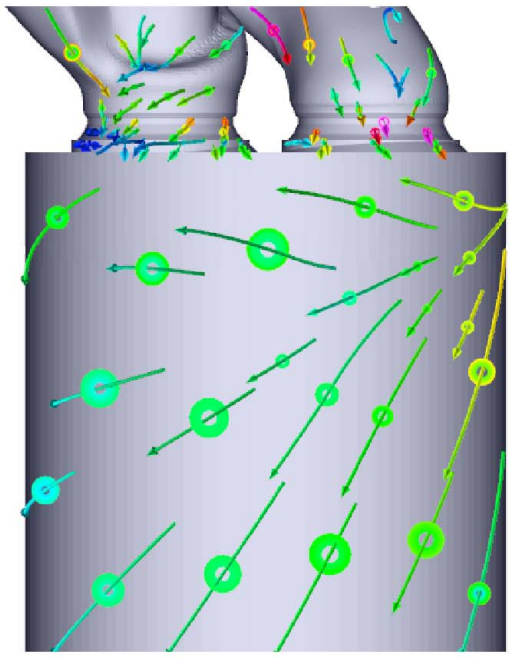


Figure 2.16: *The combination of velocity range glyphs (Disk glyph) and streamlet tubes (Arrow glyph) is applied to provide both detailed (the range glyph) and summary (the streamlet) information of the vector field direction. Image courtesy of R.S.Laramee et al. [PGL*12].*

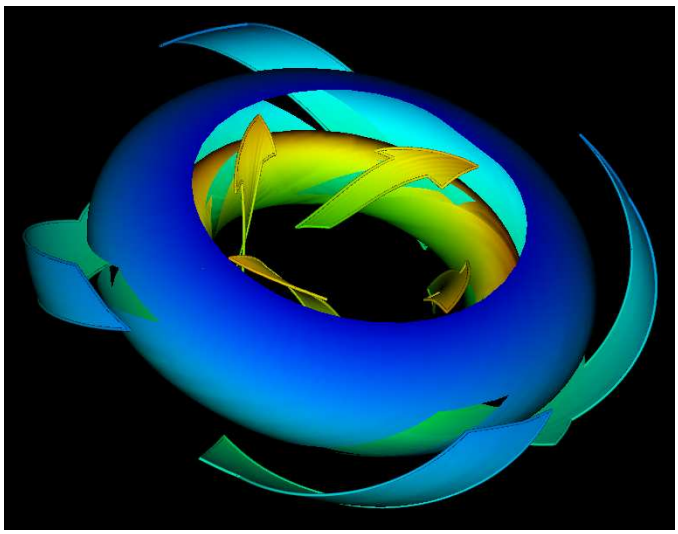


Figure 2.17: *Stream arrows offset from a stream surface. The portion offset leaves holes, reducing occlusion by the surface. Image courtesy of H.Hauser et al. [LMG97].*

3.3 Geometric Illustration Techniques

In this section we study a range of geometric illustration techniques. The purpose of illustrative techniques is to enhance the often complex surface representations commonly found in flow data. A general solution to this problem is to use transparency. With surfaces we have additional options. Surface primitives have well defined normals thus they can offer perceptual advantages including: lighting and shading which provide intuitive depth cues, the ability to texture map, image space techniques such as silhouette edge highlighting and the placement of additional geometry on the surface.

Löffelmann et al. [LMGP97] introduce methods for placing arrow images on a stream surface using a regular tiling, and offsetting portions of the surface, leaving arrow shaped holes which alleviate occlusion. The authors are inspired by Shaw's [AS92] artistic approach to the use of stream surfaces to help with occlusion within the dynamical systems. The authors extend this work as their method provides unsatisfactory results in areas of divergence and convergence [LMG97]. The arrows could become too small or too large to provide quality stream surface visualisations. The proposed novel enhancement is a hierarchical approach which stores a stack of scaled stream arrows as textures applying the suitably scaled arrow to the surfaces maintaining consistent proportionality. See Figure 2.17.

Weiskopf and Hauser [WH06] explore a Graphics Processing Unit (GPU) based shading method for the evaluation and visualisation of surface shapes. Cycle shading and hatched cycle shading methods extend the idea of natural surface highlights in a regular repeating pattern. This technique can be used where Phong illumination is appropriate as curvature or mesh connectivity information is not required. To reduce depth induced aliasing an approach similar to Mip Mapping is introduced. The effectiveness

of cycle shading for the assessment of surface quality is demonstrated by a user study.

Image-based streamline seeding on surfaces by Spencer et al. [SLCZ09] utilise the perceptual properties of surfaces by displaying the local vector field as as evenly spaced streamlines [JL97], while maintaining lighting, shading and other useful depth cues. This image-based approach is efficient, handles complex data formats, and is fast. The evenly spaced streamlines are produced by scanning the image, checking the depth buffer value of the fragments, and initialising the seeding where the fragment depth is non zero (i.e., the fragment is part of some object in the field of view) and no other streamline is closer than the user specified minimum distance.

The illustration of stream surfaces is explored by Born et al. [BWF*10] who are inspired by traditional flow illustrations drawn by Dallmann, Abraham and Shaw in the early 1980's. The authors describe techniques for silhouettes and feature lines, halftoning, illustrative surface streamlines, cuts, and slabs to visually describe the surface shape. User interactive exploration with these techniques allows insight into the inner flow structures of the data. Implemented on the GPU, this work requires no preprocessing for the final visualisations.

Further exploration of illustration techniques is performed by Hummel et al. [HGH*10] who present a novel application of non photorealistic rendering methods to the visualisation of integral surfaces. See Figure 2.18. The paper examines how transparency and texturing techniques can be applied to surface geometry to enhance the users perception of shape and direction. They describe angle, normal variation, window and silhouette transparencies with adaptive pattern texturing. The authors present this work

Figure 2.18: *Illustrative techniques applied to a stream surface. This technique shows windowed transparency and two sided surface colouring. Image courtesy of C.Garth et al. [HGH*10].*

in a combined rendering pipeline implemented on the GPU. This maintains interactivity and removes the need for expensive surface processing to generate the visualisations.

3.4 Texture-based Techniques

In this subsection we present an overview of texture-based visualisation algorithms. This work is subdivided into *Static Texture* and *Dynamic Texture* techniques. Dense texture-based techniques exploit textures to display a representation of the flow and avoid the seeding challenges. The general approach uses a texture with a filtered noise pattern which is smeared and stretched according to the local velocity field. Texture-based approaches provide dense visualisation results, show intricate detail, and capture the characteristics of the flow even in complex areas of flow such as vortices, sources, and sinks.

With standard spot noise, a texture [vW91] can be characterised by a scalar function $f(x)$. A spot noise texture is defined as:

$$f(x) = \sum a_i h(x - x_i) \tag{2.9}$$

in which $h(x)$ is called the spot function. It is a function everywhere zero except for an area that is small compared to the texture size. a_i is a random scaling factor with a zero mean, x_i is a random position. In non mathematical terms: spots of random intensity are drawn and blended together on random positions on a plane.

Static Texture

The concept of the static texture in the context of dense texture-based methods refers to the non animation of the flow in the final visualisation. Although the images are static in nature the algorithms can be applied to both steady state or time dependent flow data. One of the early examples of this classification of techniques is presented by de Leeuw and van Wijk, who first use the basic textured spot noise principle for visualising vector fields on surfaces and steady flow [dLvW95]. The authors build on work by van Wijk [vW91] extending it in four main areas; using a parametrised stream surface to deform the spot polygon adapting it to the shape of the local velocity field, using a negative Gaussian high pass filter to remove the low frequency components of the spot noise textures, using the graphics hardware to conduct a series of transformations of the matrix stack, normally used for the viewing pipeline, for improved interactive animation, and synthesising spot noise on highly irregular grids by predistorting the spots in texture space.

Extending the original LIC (Line Integral Convolution) method [CL93] to the visualisation of flows on curvilinear surfaces, Forssell and Cohen [FC95] introduce a technique to visualise vector magnitude (velocity) as variable speed flow animation. In order to extend LIC to a parametrised surface the surface geometry and related vector field information is mapped to parameter space. The LIC image in parameter space is computed

and mapped back onto the physical surface through an inverse mapping. The new visualisation takes into account the vector magnitudes by varying the speed of the phase shift in the animation, and handles unsteady flow by conducting convolution along the pathlines instead of streamlines. However, the direct convolution following the pathlines of the particles can lead to visual ambiguity since the forward and backward pathlines through a pixel need not match.

Mao et al. [MKFI97] extend the original LIC method by applying it to surfaces represented by arbitrary grids in 3D. Former LIC methods targeted at surfaces were restricted to structured grids [For94], [FC95], [SJM96]. Also, mapping a computed 2D LIC texture to a curvilinear grid may introduce distortions in the texture. The authors propose solutions to overcome these limitations. The principle behind their algorithm relies on solid texturing [Pea85]. The convolution of a 3D white noise image, with filter kernels defined along the local streamlines, is performed only at visible ray surface intersections.

Battke et al. [BSH97] introduce a fast LIC technique for surfaces in 3D space. Instead of using 2D global parametrisation, which is limited to curvilinear surfaces, the authors propose a 3D local parametrisation scheme. In this way multiple interconnected surfaces can be handled. An initial tessellation of the surface is conducted and local Euclidean texture coordinates are defined for each triangle. LIC textures are then computed by projecting a locally scaled vector field onto each planar triangle. To ensure a smooth transition across the tessellated surface, streamlines are followed across neighbouring triangles, which results in a smooth textured surface. The textures are packed efficiently into texture memory by arranging similar patches in rows and then proceeding using a greedy algorithm. The result is a smooth interactive visualisation of LIC on surfaces in 3D flow.

Shen et al. [SK98] further improve [FC95] by elimination of the artefacts caused by the ambiguity of backward and forward convolution of flow pathlines. This enables the visualisation of unsteady flow on surfaces. In order to achieve this a time accurate scattering scheme, compared to the gathering scheme in the original LIC, is used to model the texture advection. More specifically, every pixel in the image space "scatters" its colour value to its neighbours following the pathline of the particle at the centre of the pixel in a small time step. The resulting colour for each pixel is the weighted sum of all the contributions from its neighbours by considering their ages. In order to maintain the temporal coherence, this method uses the previous result as input for the next iteration. To improve the performance a parallel framework is also discussed.

Weiskopf [Wei07] introduces iterative twofold convolution as an efficient high quality two stage filtering method for dense texture-based vector field visualisation. The first stage applies a Lagrangian particle tracing-based user specified compact filter kernel. The second stage applies iterative alpha blending for large scale exponential filtering. A discussion of sampling rates demonstrates this order of convolution operations facilitates large integration step sizes. Twofold convolution can be applied to steady and unsteady vector fields, dye and noise advection, and surfaces. This work has the potential to be

incorporated in existing GPU-based 3D vector field visualisation methods.

Palacios et al. [PZ10] advance techniques which illustrate flow on surfaces. The authors present an algorithm for the visualisation of N way rotationally symmetric fields ($N-RoSy$) on 2D planes and surfaces. The basic idea is to decompose the $N-RoSy$ field into multiple distinct vector fields. Together these decomposed vector fields capture all N directions at each point. The LIC method is then adopted to compute a flow image for each vector field. These flow images are blended to obtain the final result. A simple probability model-based on the correction of normally distributed random variables is applied to compensate for the loss of contrast caused by the blending of images. This algorithm is easily extended to surfaces with an additional transformation to combat the artefacts caused by the perspective difference of the N direction vectors on the tangent plane and the view plane, respectively.

Dynamic Texture

Dynamic textures in the context of dense texture-based methods refers to the animation of the flow in the final visualisation. The technique generally known as texture advection is applied to steady state and time dependent flow data. The main challenges of the literature addressed here are computational time and visualisation of unsteady flow.

Laramee et al. [LJH03] and van Wijk [vW03] present new methods for the synthesis of dense textures on surfaces, bringing the concept into the realms of interactivity. These techniques follow previous work in this area in which van Wijk presents a method for producing animated textures for the visualisation of 2D vector fields [vW02] along

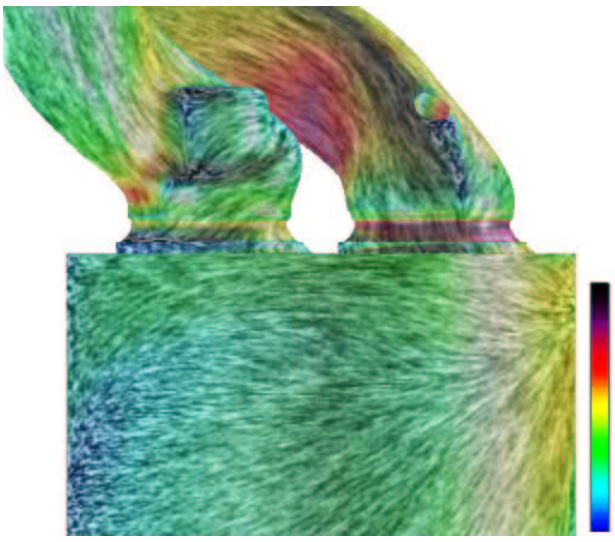


Figure 2.19: *A side view of the surface of a 221K polygonal intake port mesh. The visualisation shows ISA applied to the flow simulation data. Colour is mapped to velocity. Image courtesy of R.S.Laramee et al. [LvWJH04].*

with methods by Jobard et al. [JEH01]. This technique uses a different order of visualisation operations than traditional work. The surface geometry and related vector field information are projected to image space where the texturing is then applied.

Following this work, Laramee et al. [LvWJH04] perform a comparative analysis of two techniques: Image Space Advection (ISA) [LJH03], and Image-Based Flow Visualisation for Surfaces (IBFVS) [vW03]. The authors discuss the best application of each technique, explaining that IBFVS is a good choice where the pixel to polygon ratio in image space is high, and the ISA technique is better for larger meshes in which many polygons may cover a single pixel or where there are many occluded polygons, see Figure 2.19.

Laramee et al. [LSH04] then extend current texture advection techniques to present a novel hybrid method in which a dense texture-based flow representation is applied directly to isosurfaces. The authors build on work by van Wijk et al. [vW03] and Laramee et al. [LJH03] whose methods are suitable for the visualisation of unsteady flow on surfaces. This paper addresses issues associated with applying texture advection to isosurfaces where there may be a component of the velocity that is normal to the surface, the perceptual challenges such as occlusion, and issues related to resampling of the 3D vector field to 2D image space. The authors use a normal mask to dim areas of flow which have strong cross flow components at the isosurface, and use a clipping plane to remove subsets of the geometry to help reduce occlusion.

The proposed texture advection technique by Weiskopf and Ertl [WE04] depends on Lagrangian particle tracing, which is simultaneously computed in object space and in image space. This approach builds on previous texture advection work, introducing frame to frame coherence when the camera position is changed. The input noise is modelled as a 3D texture which is scaled appropriately in object space. The authors propose different colour schemes to improve the visualisation of shape and flow. This technique is implemented on the GPU and supports interactive visualisation.

Studying the application of image-based techniques, Laramee et al. [LWSH04] visualise the flow characteristics of the cycle of an engine cylinder, focusing on swirl and tumble motions. A variety of techniques are demonstrated including isosurfaces enhanced with an image-based technique. As another application of visualisation of automotive CFD simulations, Laramee et al. [LGD*05] analyse fluid flow within an engine cooling jacket. Again, a variety of visualisation methods are demonstrated including surface-based methods such as stream surfaces and isosurfaces.

Presenting a new extension to previous work by Laramee et al. [LSH04] and Garth et al. [GTS*04], Laramee et al. [LGSH06] propose a hybrid method where a dense texture-based flow representation is applied directly to stream surfaces. This conveys features of the flow that otherwise would not be seen using stream surfaces alone. Texture advection applied to stream surfaces avoids issues inherent with isosurfaces such as flow normal to the surface.

Modifying the texture advection approach by [WE04] to run on a highly parallelising GPU cluster, Bachthaler et al. [BSWE06] introduce an algorithm which is scalable

according to the number of the cluster nodes. The authors employ a sort first strategy with image space decomposition for LIC workload distribution. A sort last approach with an object space partitioning of the vector field increases the amount of available GPU memory. This work addresses the challenges of memory accesses locality caused by particle tracing, dynamic load balancing to support view changes, and combining image space and object space decomposition. For future work, parallel rendering could be extended to the projection of the surface geometry itself in order to visualise extremely large surface meshes.

Li et al. [LTWH08] propose a global texture advection and synthesis method for flow visualisation on surfaces. It solves the problem of inconsistent texture correspondence between visible and invisible parts of surfaces in image-based approaches. In order to achieve such global continuous texturing of flow, surfaces are firstly segmented into patches and parametrised. These patches are then overlapped by a small extent with adjacent ones and packed into a uniform texture space. Next, a 2D dense texture-based flow visualisation technique [LHD*04] such as Graphics Processing Unit Line Integral Convolution (GPULIC) or Unsteady Flow Advection Convolution (UFAC) is employed to synthesise the flow texture in texture space. The overlapping patches guarantee texture continuity across their discontinuous borders. The synthetic flow texture is mapped to the surface, compositing the overlapping regions to avoid artefacts in texture patterns caused by inconsistent partial particle traces.

4 Discussion and Analysis

No individual approach provides techniques for all problems or phenomena. The most suitable technique depends on several factors such as the purpose of the visualisation; presentation, detailed analysis, or exploration, and the interest of the analyst or engineer studying the data. This survey provides a study of a variety of techniques and approaches to cater for most eventualities when studying 3D vector field simulations. However, there are some topics of study which could potentially benefit from further examination, experiment, and verification. We summarise and discuss these challenges. Some of these challenges are more general, while others are specific to this survey.

- Visualising Error and Uncertainty.
- Implicit/Topological/Direct Techniques for Unsteady Flow.
- Interactive Construction and Rendering of Time surfaces.
- Large, Unstructured, Time Dependent CFD Data.
- Perceptual Challenges.
- Information Content.
- Human Centred Evaluation of Flow Visualisation Techniques.

Visualising Error and Uncertainty: The numerical integration of particle trajectories sampled using a piecewise interpolation of the underlying data introduces an accumulative error. This error is often overlooked or misunderstood [Hul92] described in section 1.1. Another source of error is the sampling density of the advancing front. If an insufficient density is maintained then the surface may fold or become excessively coarse producing inaccurate surface representations. The difficulty in determining the correct splitting conditions, allowing the surface trajectory to split around some boundary or critical point, is also a potential source of inaccurate representations of the flow. Distortion of the meshing techniques as described by McLoughlin at al. [MLZ10], [MLZ09], regarding distortion of non planar quads, distortion of triangles in highly diverging flow regions or areas of high shear strain between adjacent time lines according to Berger et al. [BFTW09], and the visual representation of the point strategy by Schafhitzel et al. [STWE07], are areas of future work which should be examined. The current visualisations computed with these error prone methods could lead to misleading information. Therefore, a possible future work path is to develop proper techniques to visualise these different types of uncertainty.

Implicit/Topological/Direct Techniques for Unsteady Flow: The work by van Wijk [vW93], Westermann et al. [WJE00], and van Gelder [Gel01] focuses on implicitly visualising some scalar function of the underlying flow field. Section 2.4 provides more detail. This approach avoids the numerical integration error, refinement schemes, and user defined thresholds and parameters. However, these techniques do raise new challenges such as higher dimensionality e.g., temporal data, and resolving areas of highly turbulent flow are problems which still remain unsolved.

The work by Theisel et al. [TWHS03] and Weinkauf et al. [WTHS04] use topological constructs to generate visualisations of separatrices, and singularities within the domain. These techniques are applied only to steady state flow data. Theisel et al. [TSW*05] make the step into temporal data. Ferstl et al. [FBTW10] describe fuzzy ridge structures undergoing frequent topology changes with the FTLE in turbulent areas of flow, which may cause visual clutter, providing scope for future work.

The direct methods by Peng et al. [PL08] [PGL*12] deal with unstructured meshes, with challenges arising from both the resampling computational performance and perceptual issues. Future work includes the investigation of different measures for the derivation of mesh resolution, and there is great scope to extend this work to temporal data.

Interactive Construction and Rendering of Time surfaces: There has been little work studying the challenge of time surfaces in time dependent flow with the exception of Krishnan et al. [KGJ09] (Section 2.3). This work handles the test cases well, maintaining a well formed mesh. A study of extending this work to scalable parallel environments, demonstrated on a wider range of flow types e.g., highly rotational flow,

could be conducted with a focus on representation error compared to the ground truth case.

Born et al. [BWF*10] and Hummel et al. [HGH*10] extend the use of image space and GPU technologies to significantly enhance the geometric structure of surfaces, while representing attributes of the vector field to improve the perception of flow characteristics. Extending these algorithms to dynamic surfaces such as streak and time surfaces is an area of work yet to be studied in depth. Visualising flow characteristics on the constantly changing surface geometry, which is not necessarily tangential with the underlying vector field, is one such challenge.

Large, Unstructured, Time Dependent Grids: A significant effort has gone into the study of processing large, unstructured data [USM96] [SBH*01] while more recently the focus has shifted towards studying highly parallel GPU-based implementations along with utilisation of scalable GPU-based technologies [BSWE06]. With the introduction of advanced GPU technologies an examination of techniques with a focus on parallelisation has occurred with the introduction of new challenges. With modifications required to particle tracing methods [STWE07], restrictions regarding triangulation techniques [BFTW09], and scalability are all important topics for further parallel/GPU-based research.

Perceptual Challenges: Rendering too many surfaces causes perceptual problems such as occlusion and visual complexity. Garth et al. [GTS*04] showed the placement of stream surfaces is important in the reduction of visual clutter. See section 2 for more on this. Effective and efficient placement strategies not only for static, but for dynamic data are areas of work which require significant study. The topological constructs available in steady flow can easily disappear in dynamic flow. Therefore, a consideration of the time period would likely be necessary. The work on topology aware seeding is an excellent starting point along this direction [PS09, SRWS10]. One possible next goal for this area of research to develop a knowledge assisted seeding strategy for better extracting more informative integral surfaces.

Information Content: Mapping glyphs to surfaces annotating some feature of the flow, or enabling users to switch between different visual cues, are interesting directions of future work [BWF*10]. For example, Palacios et al. [PZ10] highlight possible future work investigating efficient contrast adjustment when LIC noise images are not grey scale and have different hues. One example of this is to visualise both the major and minor eigenvector fields of a second order flow tensor. Also of interest to the authors are new decomposition strategies that may lead to fewer images to blend, thus increasing the interactivity.

A largely unexplored area of further research is the effective visualisation of multivariate data attributes normally associated with engineering simulations. This includes

not only the visualisation of the attributes themselves, but also the effective placement of the surfaces to best represent the information of interest.

Human Centred Evaluation of Flow Visualisation Techniques: Laidlaw et al. conduct an extensive user study of 2D visualisation techniques discussing their relative merits for visualising particular characteristics of the flow [LKJ*05]. Further two dimensional user studies have been conducted more recently by Liu et al. [LCS*12]. These works would not necessarily translate directly to three dimensions.

One of the key areas of future work which generally has received little attention is a thorough study of 3D visualisation techniques, outlining a taxonomy of which technique is best under a given circumstance for providing the required visual information. The work by Forsberg et al. [FCL09] is a step in the right direction performing a user study of 3D flow visualisation examining line and tube representations of integral curves with both monoscopic and stereoscopic viewing.

This type of study performed for surface-based flow visualisation examining the effective construction, placement, and rendering of surfaces to best handle, and effectively represent characteristics of a given 3D flow field is an important possible future work direction.

CHAPTER 3

Advanced, Automatic Stream Surface Seeding and Filtering

In Chapter 2 challenges such as surface placement, perception, evaluation, speed of computation, and memory footprint are highlighted as future work directions. The algorithm presented in this chapter studies a placement strategy for stream surfaces, seeding them at the domain boundary (the spatial boundary of the velocity field) and integrating them upstream through the domain. Proposals to alleviate the perceptual challenge of occlusion are studied in this chapter. The question of using streamlines and their seeding strategies for surface placement is also discussed.

A *streamline* is the trace of a massless particle from an initial location (seed point) and is always tangent to the velocity field at every point along its length. If $\mathbf{v}(\mathbf{p})$ is a 3D vector field, the streamline through a point $\mathbf{p}_0$ is the solution $I(\mathbf{p}_0,t)$ to the differential equation:

$$\frac{d}{dt}I(\mathbf{p}_0,t) = \mathbf{v}(I(\mathbf{p}_0,t)) \tag{3.1}$$

where, in the case of a steady state flow field, t is time. The initial condition is $I(\mathbf{p}_0,0) = \mathbf{p}_0$. The case of a static flow field is treated the same [GKT*08]. Streamlines are an intuitive, fast, and simple method for visualising flow. Streamlines require less computation than surfaces and are generally easier to implement than their surface counterparts. These types of curves can present disadvantages, however, such as visual clutter (when too many streamlines are rendered) and lack of depth perception.

There is no native support for the lighting of line primitives in Graphics Libraries such as OpenGL, due to the fact that line primitives have no unique normal vector. Zöckler et al. introduce a method of illuminating streamlines [ZSH96]. For the placement of

streamlines a stochastic seeding algorithm is applied. See Weinkauf et al. [WHN*03] [WT02] for applications of this seeding strategy. Mattausch et al. [MT*03] combine the illuminated streamlines technique of [ZSH96] with an extension of the evenly spaced streamlines seeding strategy of Jobard and Lefer [JL97] to 3D. The approach by Vilanova et al. [VBvP04] concentrates on seeding hyper streamlines and the use of surfaces to visualise Diffusion Tensor Imaging data. Chen et al. [CCK07] present a novel method for the placement of streamlines that does not rely solely on density placement or feature extraction. This approach is based on a similarity method which compares candidate streamlines based on their shape and direction as well as their Euclidean distance from one another.

Li et al. [LS07] present a streamline placement strategy for 3D vector fields. This is the only approach of its kind where an image-based seeding strategy is used for 3D flow visualisation. Interactive seeding strategies have been used in various modern, real world applications including the investigation and visualisation of engine simulation data [Lar02] [LWSH04] [LGD*05]. An image space-based method for placement of evenly spaced streamlines on boundary surfaces is presented by Spencer et al. [SLCZ09]. The vector field is projected onto the image plane. Thus, the complexity of tracing in the large unstructured grids that typically result from CFD simulations is avoided. Streamline density is controlled by an adaptation of the method of [JL97]. More recently Marchesin et al. [MCHM10] present a view dependent strategy for seeding streamlines in 3D vector fields. No distribution of streamlines is ideal for all viewpoints. Therefore, this method produces a set of streamlines tailored to the current viewpoint.

Methods described for seeding streamlines do not necessarily translate directly to seeding stream surfaces. Construction from discontinuous seed locations will produce surfaces which are inconsistent, twisted, folded, and self occluded, conveying little meaningful information. To produce well formed surface representations the seeding structure should be smooth; designed to represent some underlying flow characteristic. A stream surface can be described as the union of all streamlines passing through a seeding curve. It can be approximated by generating a series of streamlines along a seeding curve and joining them to produce a polygonal representation. Chapter 2 Section 2.3 discusses stream surface construction techniques in greater detail.

Stream surfaces are useful for understanding flow structures within a single time step or static flow field and are relatively simple to compute. Stream surfaces for visualisation face many challenges. These surfaces must represent an accurate approximation of the underlying simulation. Adequate sampling must be maintained while reducing the unnecessary computational overhead associated with over sampling. The problem of occlusion may stem from multiple surfaces that occlude one another, a large surface that results in occluding itself, or a combination of both.

While surfaces offer many advantages in terms of information content, a basic visualisation of the surface alone may not provide sufficient information about the underlying data. For example a stream surface alone does not show the behaviour of any inner

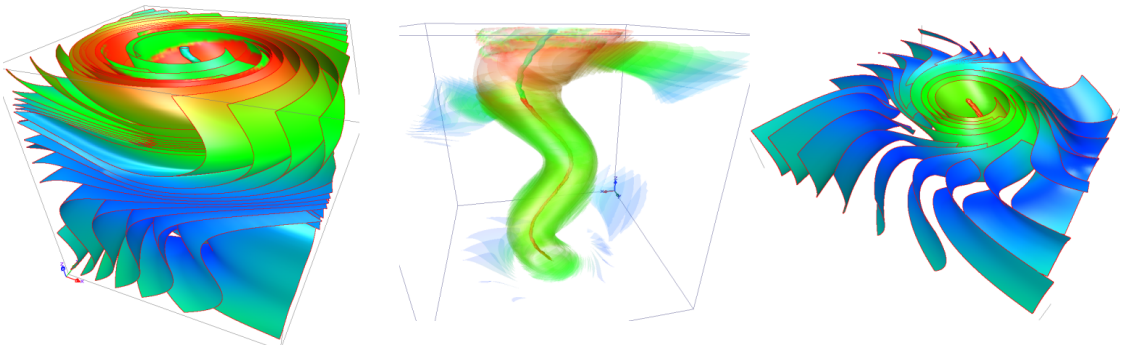


Figure 3.1: *Automatic Stream Surface Seeding: A set of stream surfaces seeded automatically using our technique on the* 128^3 *tornado simulation at time step zero. The left image shows surfaces visualised using silhouette edges. The centre image shows surfaces with opacity mapped to the magnitude of local curl. Pixels are filtered according to opacity. This reduces occlusion and allows insight into the inner flow structures. The right image demonstrates the use of clipping planes.*

flow contained within the surface. Illustrative techniques can be used to improve perception and information content. Surface primitives (as opposed to curves in space) have well defined normals. They offer perceptual advantages including: lighting and shading which provide intuitive depth cues, the ability to texture map including texture advection [LGSH06], the placement of additional geometry on the surface [LMGP97], and their use for depicting boundaries. Utilising depth cues for depicting surface shape, colour mapping and texturing to convey broader/clearer information content to the user, are examples of the advantages surface visualisations can provide. Examples of illustration techniques can be seen in Chapter 2 Section 3.

A significant body of research has been invested into automatic seeding strategies using streamlines, but, little has been offered for automatic stream surface seeding. Manual seeding is the most common method for the placement of stream surfaces. However interactive stream surface placement is based on trial and error. Important characteristics of the flow can easily be missed. To maximise the information content for the user, stream surfaces must be carefully illustrated and seeded such that they capture the underlying characteristics and features of the flow. Surfaces generally suffer from less visual clutter than lines, points, or other geometric primitives because they offer greater spatial continuity. Stream surfaces are an effective medium to convey not only the characteristics of the flow structures, but can also communicate additional information to the user. This provides strong motivation for studying stream surfaces and their seeding. The benefits and contributions of this chapter are:

- A novel approach to seeding stream surfaces in 3D flow fields automatically seeds surfaces throughout the domain based on user specified separation.
- Effective boundary computed seeding curve prioritisation.

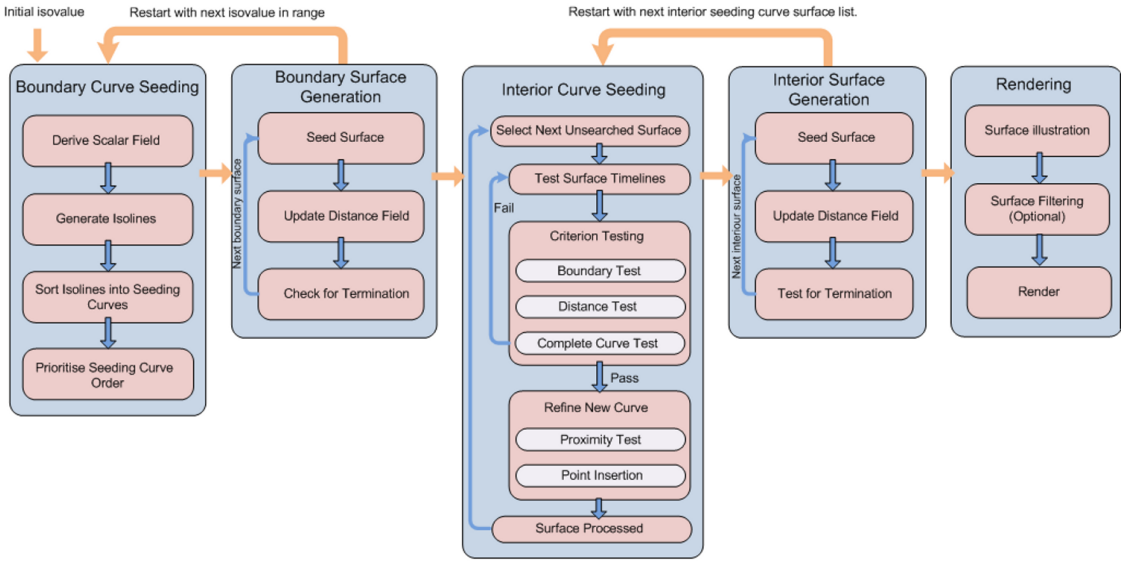


Figure 3.2: *Algorithm pipeline: The first stage creates a scalar field at the domain boundary Ω' based on the angle of incidence of out flow. A set of isolines is generated, then prioritised to create seeding curves for the stream surfaces. The stream surfaces are integrated through the velocity field. In parallel with surface advancement, a distance field, Ω_d, is updated describing the proximity to existing surfaces. This process is repeated for a range of isovalues. After generating an initial set of surfaces, new seeding curves are computed and refined in the domain interior Ω. Each surface, in a list of unprocessed surfaces, is searched along its length to locate timelines for the potential production of seeding curves. Timelines are offset at distance d_{sep} in the direction of the surface normal. If they pass a set of refinement criterion, they are used for the construction of smooth seeding curves. These seeding curves initialise new stream surfaces. This process is repeated until Ω is covered. The final stage filters and renders the surfaces using semi opacity based on curvature, clipping, and colour mapping.*

- Surface to surface proximity termination based on distance field techniques.
- Techniques for surface filtering and illustration, enhancing the information content.

A detailed presentation of the algorithm is given in Section 2. The results are discussed in Section 3.

2 Automatic Surface Seeding

This section presents the automatic stream surface seeding algorithm. We start with an overview of the seeding pipeline. Refer to Figure 3.2. The achievement of complete domain coverage in areas such as recirculating flow is the focus of the seeding strategy. Generation of surfaces in areas of flow which cannot be traced to the domain boundary

is important to provide a complete visualisation. The algorithm input is a 3D steady vector field $\mathbf{v}(\mathbf{p}) \in \mathbb{R}^3$ where $\mathbf{v}(\mathbf{p}) = [\mathbf{v}_x(x,y,z), \mathbf{v}_y(x,y,z), \mathbf{v}_z(x,y,z)]$ defined for $\mathbf{p} \in \Omega$ and $\Omega \subset \mathbb{R}^3$

1. The starting point is the derivation of boundary seeding curves. We generate the seeding curves from isolines derived at the boundary of Ω denoted Ω'. The 2D scalar field $f(\Omega')$ is a function of flow direction exiting Ω. The isolines are constructed using marching squares. Once the vertices are correctly ordered and stored, the seeding curves are then prioritised into a suitable seeding order. Refer to Section 2.1.
2. Once the boundary seeding curves are computed, we generate stream surfaces advancing through Ω until they meet terminating conditions. The termination parameters are maximum length l_{max} and distance d_{test} to neighbouring surfaces. A distance field is used to evaluate d_{test}. The process is repeated from step one until all isovalues are seeded. Section 2.2 and 2.3 provide further details.
3. We search the initial surface list along their length to locate empty regions. Each timeline of a given surface is offset normal to the surface d_{sep} . The proposed timeline is then analysed for suitability. If the resultant curve does not pass the suitability criterion, we recursively select the next downstream timeline. Refer to Section 2.4.
4. The next step produces a smooth seeding curve from a suitable offset timeline. This refinement is achieved by removing vertices which are in close proximity to each other. Interpolating new vertices maintains an evenly distributed smooth discretised curve. Refer to Section 2.5.
5. The originating surface is transferred to the processed list and searched no further once a suitable seeding location is found. The new interior stream surface is added to the list of unprocessed surfaces. The surfaces are searched both sides iteratively through the list of unprocessed surfaces until empty. Section 2.6 provides further details.
6. The final step is rendering the scene. A number of techniques are implemented to aid the viewer in perceiving the resulting visualisation. This includes the use of transparency, colour mapping, clipping planes, edge highlighting, and surface filtering. Refer to Section 2.7.

2.1 Boundary Seeding Curve Generation

For the problem of generating seeding curves between and inclusive of *boundary switch curves* (previously defined in Chapter 2 Section 2.5), we compute a scalar field representing the flow exit trajectory from the domain boundary. This approach reduces the problem to a simple marching squares isoline extraction technique.

Generation of seeding curves from isolines derived from Ω' is performed in three steps. The first step is to define a scalar field. The derived field $f(\Omega')$ represents the scalar field which is a function of the direction of flow exiting Ω. The scalar represents

the angle of incidence between the vector field defined at Ω' and the domain extent itself. The calculation is performed by projecting the unit vector onto a plane at Ω' (See Figure 3.3). The resultant magnitude $|\mathbf{v}'|$ is used as the scalar. If the exit trajectory is perpendicular to the domain boundary a scalar value of zero is stored, if the exit trajectory is parallel to the boundary then the scalar is stored as unity.

The next step is to construct the isolines from the scalar field Ω' using a marching squares algorithm. The resulting vertices would normally be rendered as order independent line segments. However the vertices require sequential ordering for the seeding curve. Modifying the initial isovalue used for the generation of the seeding curves produces results that are different.

The idea of using isolines derived from exit flow direction is the binding of the coherent flow structures at the boundary and tracing them through the domain. *Boundary switch curves* [WTHS04] are an example of this, where an isovalue of 1.0 equates to a *boundary switch curve*. The range of isovalues used in the construction of the isolines, and resultant density of seeded surfaces, can be specified by the user. This effectively seeds surfaces between pairs of *boundary switch curves*, where they exist. The algorithm subdivides the range from 1 to 0 by a user defined division. i.e. if the range is divided by 5, then isovalues at 0.2 steps are used (e.g. 1.0, 0.8, 0.6, etc.). The seeding curves, and then surface generation is performed iteratively from the first isovalue to the last e.g. curves at isovalue 0.8 are generated, then the surfaces for that value are constructed, then curves at the next isovalue (0.6) are constructed etc. It is important to note that the isovalue is proportional to $\sin\theta$. This produces denser surface seeding closer to *boundary switch curves* when using evenly spaced isovalues.

The seeding curves are prioritised in order of surface generation. The order of seeding new surfaces can influence the resulting visualisation. A logical order is longest

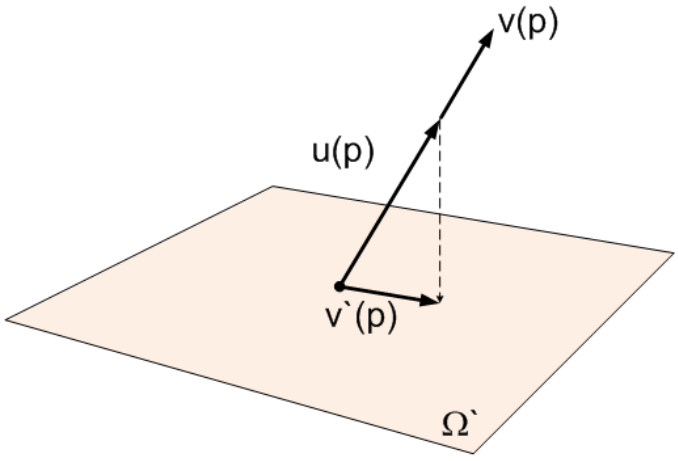


Figure 3.3: *Vector projection.* $\mathbf{u}(\mathbf{p})$ *is the unit vector of* $\mathbf{v}(\mathbf{p})$*, e.g.* $\mathbf{u}(\mathbf{p}) = \mathbf{v}(\mathbf{p})/|\mathbf{v}(\mathbf{p})|$*.* $\mathbf{v}'(\mathbf{p})$ *is the vector projected onto* Ω'*.*

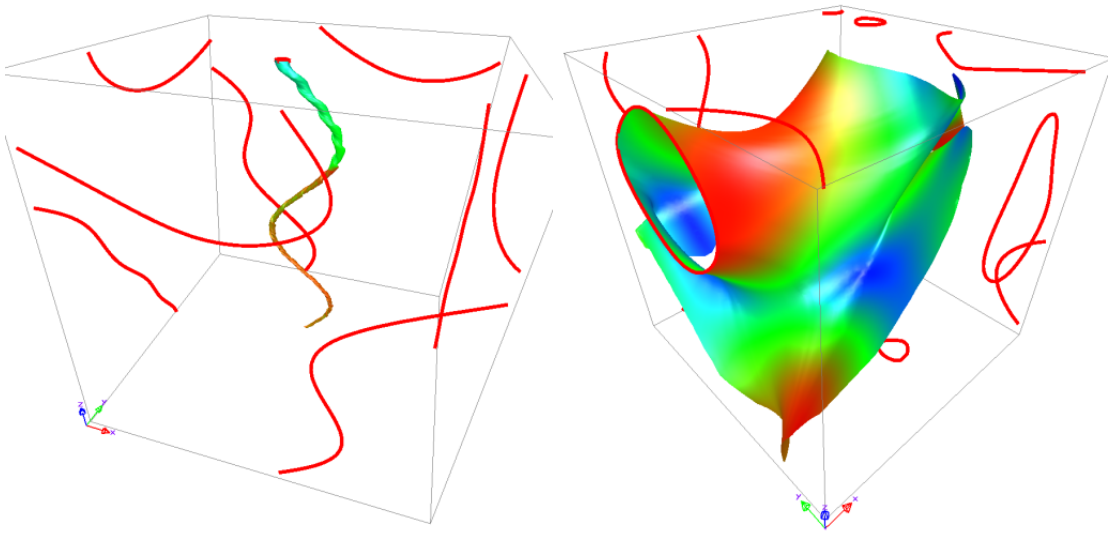


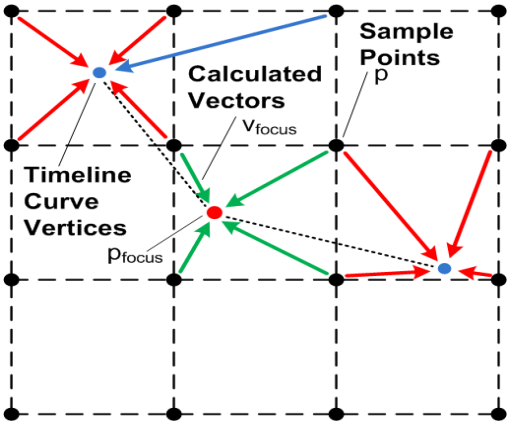
Figure 3.4: *Tornado and ABC simulations with boundary seeds generated using an isovalue of 0.9. In these figures a surface is generated from the first boundary seeding curve in the prioritised list of curves. The left image illustrates the prioritisation capturing the vortex core of the Tornado. The right image shows the longest surface propagating from a closed loop boundary seeding curve.*

seeding curve first. This heuristic is based on the idea of prioritising large domain filling surfaces. After some experimentation we find that seeding from closed loop curves is beneficial to visualising vortex cores. The final heuristic is to seed closed loop curves first; longest to shortest, and then seed open ended curves; longest to shortest, starting with an isovalue nearest one; seeding each set of isolines in descending order of isovalue. Refer to Figure 3.4.

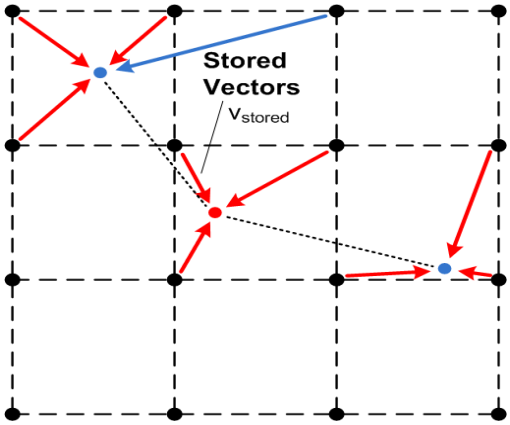
2.2 Boundary Surface Generation

Our work utilises a standard solution to stream surface computation (See [GKT*08] using the integration scheme in Appendix A). Stream surfaces are propagated from each of the seeding curves defined in section 2.1. The surfaces are then terminated according to a set of conditions such as l_{max}, boundary proximity, and d_{test}.

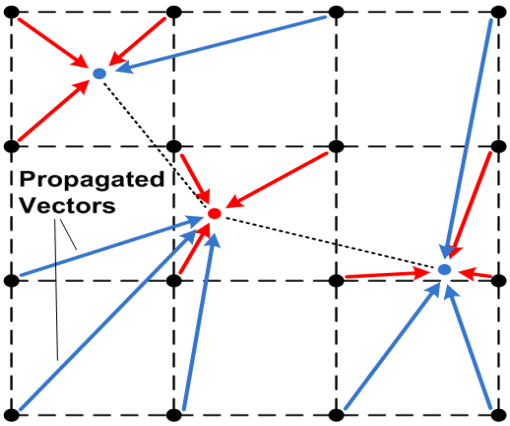
Calculating surface integration distance and determining boundary proximity are straightforward. However a distance field Ω_d is used for the efficient detection of neighbouring surfaces. As each surface is generated, its location is added to Ω_d. Then Ω_d is updated. As the next surface is propagated through Ω, it is tested against the Ω_d to determine if the proximity to any neighbouring surfaces is less than a predefined minimum distance d_{test}. If so the surface propagation is terminated. This process is repeated for all surfaces.



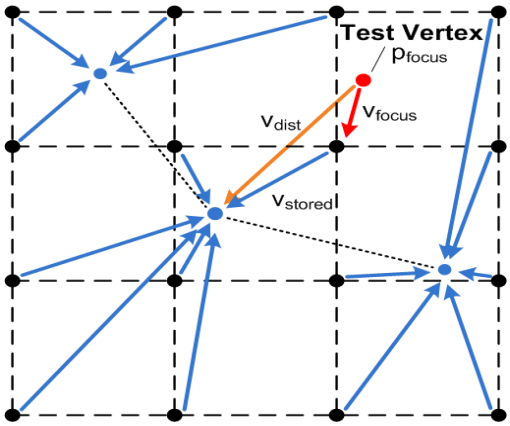
(a) For a given set of vertices where the focal vertex $\mathbf{p}_{focus}$ is highlighted in red, a set of vectors $\mathbf{v}_{focus}$ representing the distance from each of the vertices $\mathbf{p}$ of the bounding cell are calculated, i.e. $\mathbf{v}_{focus} = \mathbf{p}_{focus} - \mathbf{p}$.



(b) $|\mathbf{v}_{focus}|$ is compared to the currently stored vectors $\mathbf{v}_{stored}$, which are replaced if the magnitude is smaller, i.e $\mathbf{v}_{stored} = min(|\mathbf{v}_{stored}|, |\mathbf{v}_{focus}|)$.



(c) The vector information is then propagated throughout the domain using the Vector City Vector Distance Transform (VCVDT) method which is detailed in [SJ01].



(d) Proximity is computed thus: $|\mathbf{v}_{dist}| = |\mathbf{v}_{focus} + \mathbf{v}_{stored}|$, where $\mathbf{v}_{focus}$ is the distance from the test vertex $\mathbf{p}_{focus}$ to the cell vertex $\mathbf{p}$.

Figure 3.5: *The distance field is a fast and versatile method used to locate the nearest object in the domain. The distance test compares the shortest magnitude to a predefined minimum allowable distance.*

2.3 Distance Field

The detection and computation of distance to the nearest object in a given domain from any given point is non trivial. A brute force approach can be implemented testing the distance to every vertex of every object in the domain with the current vertex and storing the shortest. This method is very expensive and thus distance field techniques to improve the speed are used. In general, there are two groups of approaches [JBS06]:

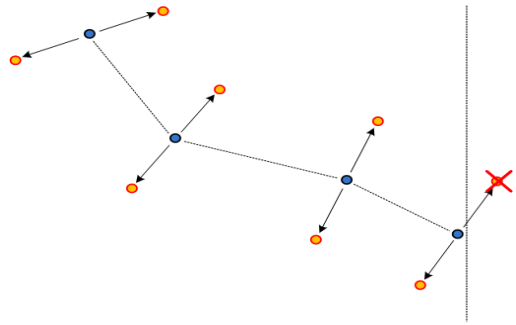
Distance computation for common surface representations, which discards most of the objects by exploitation of spatial coherency, e.g., computing distances from objects in close proximity. Distance transforms are a second category which initially use a similar technique to evaluate distances to certain regions, e.g., a thin layer around the surface, and then propagates them through Ω.

The most interesting method appropriate for use with discretised surfaces is the Vector City Vector Distance Transform or VCVDT [SJ01]. The first step is to calculate the vector $\mathbf{v}_{focus}$ from the vertex of interest $\mathbf{p}_{focus}$ (in our case surface vertices) to the bounding cell vertices $\mathbf{p}$. See Figure 3.5(a). The vector magnitude, $|\mathbf{v}_{focus}|$, is compared with the currently stored vector magnitude, $|\mathbf{v}_{stored}|$, at all bounding cell locations. The shortest in each location i.e. $|\mathbf{v}_{focus}| < |\mathbf{v}_{stored}|$ is stored replacing the original vector. See Figure 3.5(b). This process is repeated for every vertex representing the surface. Ω_d is then propagated throughout the domain. See Figure 3.5(c). While computing a new surface each new vertex is proximity tested against Ω_d. If the distance is too short $|\mathbf{v}_{dist}| < d_{test}$ then the surface front propagation is terminated. See Figure 3.5(d). Once each surface is terminated it is then added to the distance field regardless of termination method.

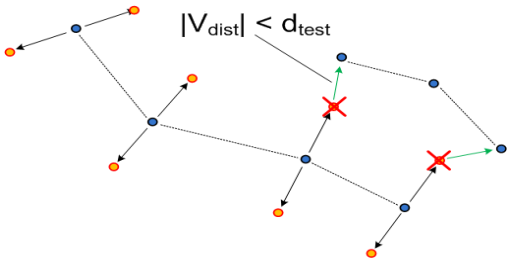
2.4 Timeline-based Seeding Curve Generation

This stage of the pipeline takes a list of stream surfaces seeded from Ω' from which additional surfaces can be seeded in order to gain complete domain coverage. This list is initially unprocessed. Each unprocessed surface is searched in turn to find a suitable location to generate a new seeding curve. This involves offsetting each timeline in turn along the length of the surface to find empty regions at which seeding of new interior stream surfaces can take place. Each timeline of a given stream surface is projected normal to the surface and analysed for suitability against predefined criteria. The surfaces are searched both along their front and rear face.

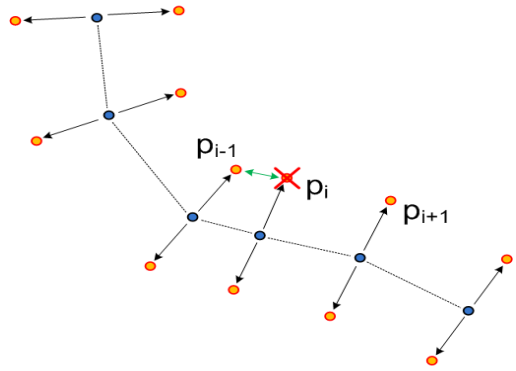
The first criteria is the boundary test. Every new vertex of the candidate seeding curve must be inside Ω. Therefore any vertex $\mathbf{p}_i$ projected outside Ω is rejected (Figure 3.6(a)). The second criteria is the distance test. Each candidate curve vertex is tested against Ω_d. The vertex is marked reject if the location is within a user defined minimum distance parameter d_{test}, e.g. too close to a neighbouring surface (Figure 3.6(b)). The candidate curve is then tested for spatial continuity guided by user defined parameters of minimum contiguous length d_{length}, and maximum split distance d_{split}. If any resulting gaps in the offset timeline are too large, or any remaining section is too short, the candidate seeding curve is rejected. If the candidate curve does not pass the criterion then the next timeline along the length of the surface is selected, and the process repeats.



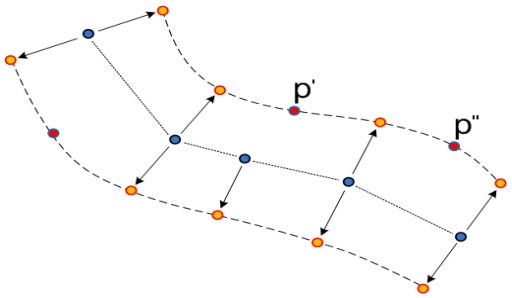
(a) The proposed seeding point is outside the domain boundary and is rejected.



(b) The two rejected proposed seed points are in too close proximity to another surface.



(c) The proposed seed points are too close and is rejected.



(d) New vertices are inserted into the seeding curves.

Figure 3.6: *The process of accepting or rejecting proposed new interior seeding points.*

2.5 Interior Seeding Curve Refinement

If the candidate interior seeding curve passes the previously described criterion it is refined. The refinement process is intended to evenly distribute and smooth out any inconsistencies with the new curve. As a result of the projection in concave or convex areas of the surface the vertices may be too close to one another, too far apart.

The refinement starts by removing vertices too close in proximity. The vertices are tested in groups of three, $\mathbf{p}_{i-1}$, $\mathbf{p}_i$, $\mathbf{p}_{i+1}$, starting from one end of the curve, incrementing one vertex at a time. The purpose of this approach is to test the central vertex against a user defined proximity d_{prox} to its neighbours. If a neighbouring vertex is too close $|\mathbf{p}_{i-1} - \mathbf{p}_i| < d_{prox}$ or $|\mathbf{p}_i - \mathbf{p}_{i+1}| < d_{prox}$, vertex $\mathbf{p}_i$ is removed (Figure 3.6(c)).

The curve is further refined by inserting additional vertices at locations where proximity to the next vertex is too great. The insertion process uses cubic interpolation

(Catmull-Rom spline). The process of insertion marks the position along the curve a new vertex $\mathbf{p}'$ should be inserted. A new vertex is interpolated and held in a temporary list. This is repeated for every section along the curve (Figure 3.6(d)).

Once completed the temporary list of vertices, $\mathbf{p}'$ and $\mathbf{p}''$ in the example, are then inserted at the correct locations along the seeding curve. The approach of inserting the vertices into the array post interpolation prevents newly inserted vertices from violating the proximity test, interfering with the interpolation. This process is repeated for all candidate interior seeding curves.

2.6 Interior Stream Surface Generation

Once the seeding curve has been formed, a new surface is generated in both downstream and upstream directions. Distance field Ω_d is updated with the vertices representing the new surface. The given stream surface is added to the list of processed surfaces. The new surface is added to the list of unprocessed surfaces. This is repeated for every unprocessed surface until no further interior seeding curves, and therefore surfaces, can be generated. The surface termination criteria are the same as for boundary surfaces.

2.7 Stream Surface Filtering and Rendering

A number of techniques are implemented to aid the viewers perception of the visualisation. The techniques used to render the results include the use of transparency, colour, silhouette edge highlighting, lighting and shadow, and surface filtering. Colour is often mapped to $|\mathbf{v}|$ and opacity is mapped to the magnitude of vector field curl $|\nabla \times \mathbf{v}|$. The curl of vector field $\mathbf{v}$ is described as a vector having magnitude equal to the circulation at each point $\mathbf{v}(\mathbf{p})$, and is perpendicular to the plane of circulation at each point. In Cartesian coordinates, this is defined as:

$$\nabla \times \mathbf{v} = \left(\frac{\partial \mathbf{v}_z}{\partial y} - \frac{\partial \mathbf{v}_y}{\partial z}\right) i + \left(\frac{\partial \mathbf{v}_x}{\partial z} - \frac{\partial \mathbf{v}_z}{\partial x}\right) j + \left(\frac{\partial \mathbf{v}_y}{\partial x} - \frac{\partial \mathbf{v}_x}{\partial y}\right) k \tag{3.2}$$

where i, j, k are components of a unit vector. Lighting and shading are standard tools for depth and shape perception. Silhouette edge highlighting is used to help the viewer understand where the surfaces curve away from any given viewpoint, and enhances the perception of surface edges.

Another technique involves filtering of the surfaces to aid in the reduction of visual clutter. As our visualisations are rendered with opacity mapped to $|\nabla \times \mathbf{v}|$, the result is opaque surfaces in areas of high vector field curvature i.e., opaque vortex cores. This criterion is defined on a normalised scale which can be interactively adjusted by the user. An example of this can be seen in Figure 3.1 (middle).

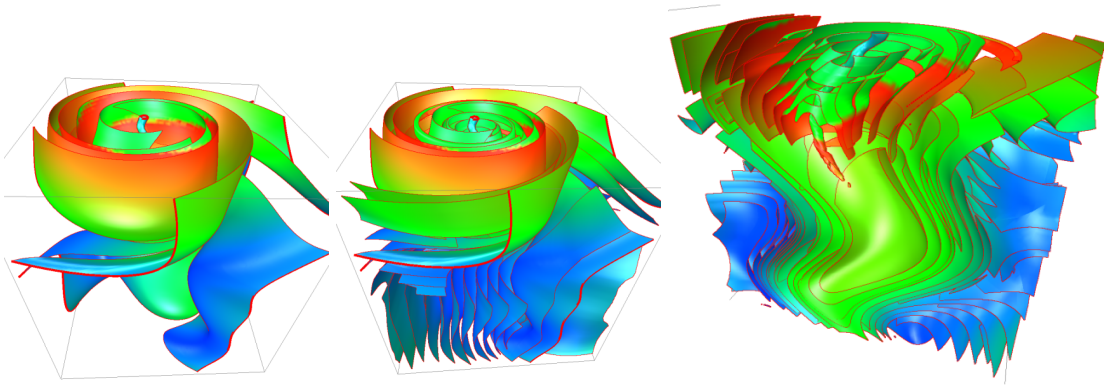


Figure 3.7: *The* 128^3 *tornado simulation. The top left image shows surfaces seeded with boundary seeding using an isovalue of 0.9. The boundary seeding curves are highlighted in thick red. The top right image shows seeding interior surfaces at* $d_{sep} = 5.0$*. The bottom image shows the final visualisation with silhouette edges and clipping planes.*

3 Results

To demonstrate the technique we choose a number of datasets which best capture the range of scenarios we may encounter when studying CFD data. The Tornado data best highlights a large vortex structure filling the domain and thus curved areas of flow crossing the domain. The cuboid data represents data with an inflow and outflow region, and flow past an object with turbulence. The Lorenz Attractor and ABC data are highly curved analytical datasets, where the curvature is formed from different phenomena, i.e., singularities (Lorenz Attractor) or waveforms (ABC).

We achieve full domain coverage and capture the properties of the flow field by allowing the user to control the density of neighbouring stream surfaces. Figure 3.1 shows results from automatic seeding in a tornado simulation. The surfaces are seeded using

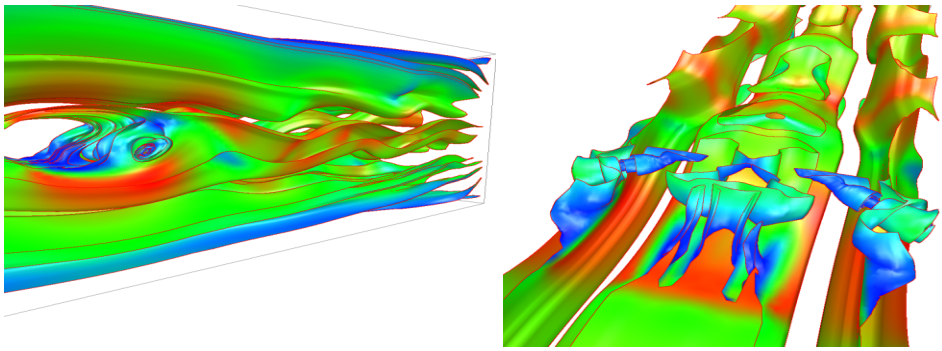


Figure 3.8: *A set of stream surfaces on a simulation of flow behind a cuboid [CSBI05] [vFWTS08a]. The images show a set of 5 isovalues used to fill the domain. Colour is mapped to velocity. The left image visualises the generated surfaces and demonstrates domain coverage. The right image uses edge highlighting and is clipped to revel the centre of the visualisation. Note: The cuboid is not shown for clarity.*

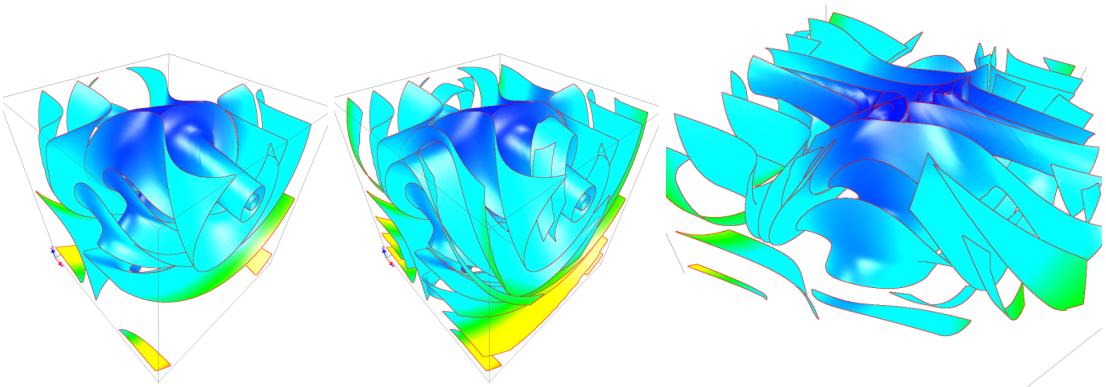


Figure 3.9: *The Lorenz attractor with a resolution of* 128^3*. The parameters for this simulation are;* $\sigma = 10.0$*,* $\rho = 28.0$*, and* $\beta = 8/3$*. The familiar circulation interaction is seen from the underside of the domain. This visualisation shows a spars set of seeding parameters. The left image shows boundary seeding at 3 intervals. The middle image shows the interior seeding filling the remaining unseeded areas of the domain using* $d_{sep} = 10.0$*. The right image is clipped revealing the inner flow characteristics.*

boundary seeding from isovalue 1.0 to 0.0, at isovalue intervals of 0.2. It can be seen that the domain is adequately covered to capture the structure of the tornado. On any planar domain boundary, the scalar field range may not extend to 1 (*boundary switch curve*). However, our method is able to generate seeding curves at isovalues less than 1 effectively extrapolating between *boundary switch curves* on an extended boundary plane. Using transparency, filtering and silhouette edges improve the users perception of the results. The centre image highlights well the tornado core using curl-based surface filtering. Figure 3.7 demonstrates the interior seeding and resultant surfaces filling the domain from an initial set of surfaces generated at the boundary. The bottom im-

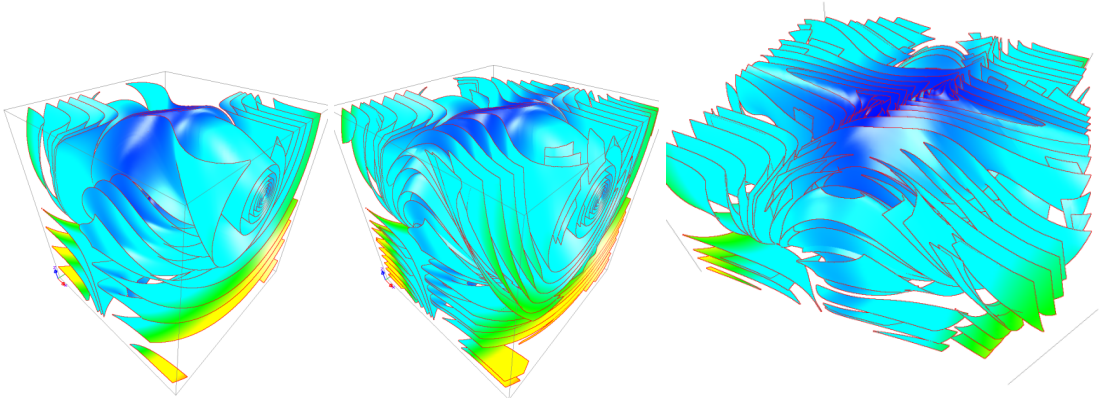


Figure 3.10: *This visualisation of the Lorenz attractor shows the effect of denser seeding. The left image shows boundary seeding at 5 intervals. The middle image shows interior seeding using* $d_{sep} = 5.0$*. The right image is clipped revealing the inner flow characteristics.*

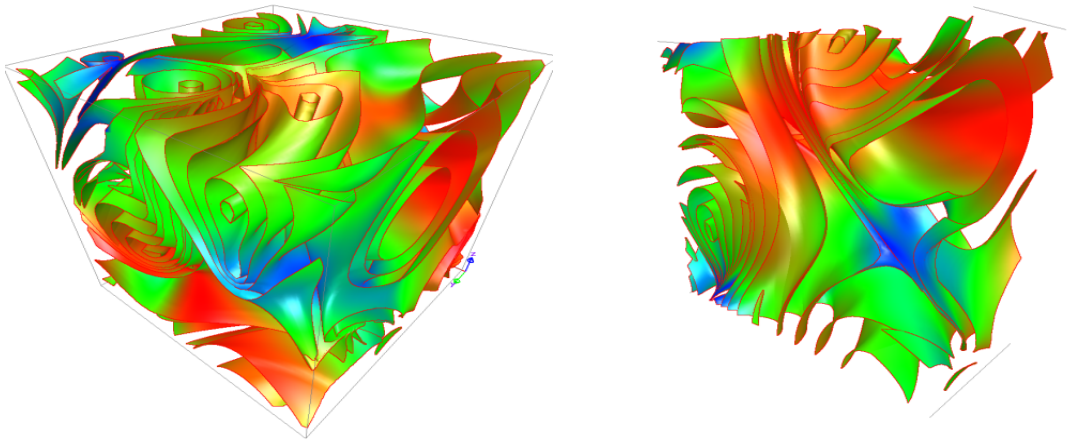


Figure 3.11: *A 128^3 Arnold Beltrami Childress (ABC) simulation. The left image shows boundary seeding at 5 intervals. The images show surfaces with colour mapped to velocity. The right visualisation is sliced using clipping planes.*

age uses a clipping plane to section the flow field. It clearly shows the interior seeding following the structure of the tornado. Figure 3.8 visualises the vortices generated behind the cuboid. A section through the visualisation aids the users perception of these characteristics.

The Lorenz attractor can be seen in Figures 3.9 and 3.10. The images in Figure 3.9 show the vector field visualised with a sparse set of surfaces. The top left image is seeded only from the boundary. Using interior seeding the top right image shows complete domain coverage. The bottom image uses clipping planes to understand the inner flow structure. The second set of images in Figure 3.10 demonstrates a denser visualisation which better captures the underlying flow characteristics using the same techniques as Figure 3.9. The ABC visualisation shown in Figure 3.11 demonstrates the capture of the inner flow structures. The use of clipping planes and edge highlighting significantly improve the perception of the flow field.

The illustrative strategies implemented for resolving clutter, resulting from seeding multiple surfaces, improve perception and therefore aid understanding of the underlying flow structures. Visualisation of less complex flow characteristics produce clear results from the different strategies employed. When visualising more complex flow data enabling the user to filter surfaces (Figure 3.1 middle) or clip areas of the domain (Figure 3.11 right) can reduce the visual clutter improving the information content of the visualisation. In practical applications indicating the flow direction would by required by the user. This can be implemented using textures applied to the surface mesh, or using techniques such as [CSFP12] by Carnecky et al. Also, stream surface generation can be influenced by topological structures such as invariance and separation features and is discussed in depth by Schneider et al. [SRWS10].

CHAPTER 4

Automatic Stream Surface Seeding: A Feature Centred Approach

CHAPTER 3 studies stream surface seeding at the domain boundary followed by interior stream surface seeded at a user specified density. This approach combined with filtering and rendering techniques improved the perception and information content of the visualisations, while reducing the visual clutter. Some challenges remain with these approaches such as; seeding surfaces where there is no boundary inflow, manual interaction for the filtering process, and realtime interaction with large datasets.

To address these challenges this chapter investigates the hypothesis that stream surfaces must be placed such that they capture the important characteristics of the flow field with minimal quantity. Seeding curves must be positioned and oriented in the neighbourhood of geometric structures best representing the flow field. Illustrative techniques must enhance the perception of surface structures supporting the roll of effective placement. Stream surface orientation is an important consideration in light of these requirements. Of the possible visualisation techniques that can be used to simplify the flow field and present potential seeding locations, vector field clustering algorithms and feature extraction techniques are considered.

Explicit feature extraction techniques perform a search of the flow field in order to locate and visualise specific subsets of the flow [PVH*03]. However, features are not always well defined. For example, there is no universal definition of a vortex. Features are often user dependent and thus they may not always be predicted a priori [LGD*05]. Each type of feature requires a special algorithm to extract it. Implementation of an algorithm for each type of feature may not be practical. Many explicit feature extrac-

tion algorithms rely on threshold values which ultimately determines the presence (or absence) of the feature of interest. This can result in false positives or missing subsets of interest (this is especially true for 3D CFD data exploration). In contrast, vector field clustering does not rely on an algorithm threshold value and can highlight weak or partial features in the flow. Vector field clustering has been used to show interesting flow features for real world datasets [PGL*12].

Peikert et al. [PS09] present topology-based methods for constructing stream surfaces in the neighbourhood of critical points. The authors discuss error bounds and give application examples for the range of topological features under consideration. Topological methods are of limited use for this application. First, no sources or sinks are present in our 3D flow simulations. Periodic orbits are quite rare and saddle points are certainly not the only features of interest to the user. Plus, explicit extraction of topology also relies on special threshold values which may result in overlooked features of interest [LHZP07]. We demonstrate how our algorithm constructs stream surface seeding curves in areas of interest such as rotational flow (fluid which rotates about an axis), high curvature (fluid with a high degree of direction change) and saddle points (an infinitesimally small location of stationary flow).

Vector field clustering algorithms provide a general approach to generating a simplified, feature-based representation of vector fields. We employ this approach in our framework of automatic placement of the seeding curves. Clustering distance measures can be tuned to yield a range of results. They offer the advantage of presenting a detailed representation in important areas of the flow field, and offer flexibility required by the user. General clustering algorithms are based on agglomerative hierarchical grouping techniques which are widely used in the information visualisation domain [XW05].

In this chapter, we present a novel adaptation of a vector field clustering algorithm that can be guided by the user in order to automatically seed stream surfaces. The clustering technique, based on a distance measure driven by error, is used to locate potential stream surface seeding positions which are used to generate the final visualisations. The main benefits and contributions of this chapter are:

- An adaptation of vector field clustering to guide stream surface seeding.
- An approach to automatically locating seeding positions associated with important structures within the 3D flow field.
- A technique for generating seeding curves automatically which reflect important characteristics of the flow field.
- A technique combining flow curvature with illustrative techniques to provide enhancements to the perception of the visualisation.

We illustrate how to capture the characteristic structures within the flow field such as rotational flow. The key to capturing the required properties of the flow is in defining the appropriate clustering metric and providing the user with the flexibility to guide the

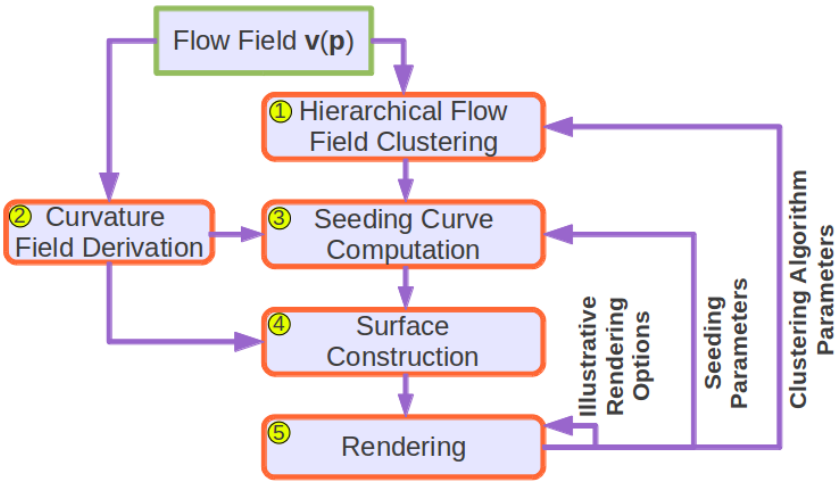


Figure 4.1: *The automated stream surface seeding pipeline. The pipeline shows the vector field clustered with customisation parameters, the derivation of the curvature field, and seeding curve generation. Stream surfaces are then propagated from the seeding curves through the vector field, and then rendered.*

seeding within the domain. Our focus pays particular attention to the flexibility of the clustering technique combined with the seeding curve generation strategies.

The rest of this chapter is divided into the following sections: A detailed presentation of the algorithm is given in Section 2. The results are reviewed in Section 3. Domain expert feedback is provided in Section 4.

2 Automated Stream Surface Seeding

This section describes the stream surface seeding algorithm for the visualisation of vector fields. We start with an overview of the processing pipeline illustrated in Figure 4.1. The algorithm is presented in multiple stages consisting of the clustering process, calculation of the curvature field used to support seeding and illustration, seeding curve generation, stream surface generation, and rendering. The choice of clustering for generating good seed locations is based on the premise that the clustering process can guide seeding, at varying levels of simplification, in the vicinity of the flow features of interest to the user. Examples of this can be seen in Section 3. Domain expert feedback can be found in Section 4. This process results in stream surfaces which represent the flow structures contained within the domain.

1. Clustering is performed on the flow field with options that guide the clustering process. The parameters provide the user with great flexibility in controlling the density of surfaces and prioritising the formation of clusters at locations corresponding to subsets of the flow interesting to the user. Vector direction, magnitude, and location are comparison attributes that can be set. See Section 2.1.

2. The curvature field is derived from the flow field. It is used to compute seeding curves after the clustering process has identified the seeding locations. It is also used to map opacity to stream surfaces for illustrative rendering. See Section 2.2.

3. The error level parameter, or simplification level s_l, controls which clusters are selected from the hierarchical tree providing the required level of detail or seeding density. Seeding curve generation starts with the selection of representative clusters. The cluster location is automatically used as the basis of the seeding curve. The seeding curves are then generated by integrating forward and backward through the curvature field at a length proportional to the cluster size. This generates seeding curves which follow the local flow structures while maintaining orthogonality with the flow. See Section 2.3.

4. Once the seeding curves are computed, stream surfaces are propagated from each of the seeding curves. Flow attributes such as velocity magnitude may be colour mapped. Opacity can be mapped to local flow curvature. See Section 2.4.

5. After the generation of the stream surfaces, the surface data is rendered using a number of illustrative techniques to enhance the perception of the flow field. Opacity derived from curvature of the flow field is utilised in conjunction with depth peeling for rendering semi transparent surfaces. See Section 2.5.

2.1 Customised Clustering

The algorithm presented in this chapter adapts the clustering method of Telea and Van Wijk [TvW99]. The aim of this work is to generate a globally simplified vector representation while maintaining enough detail to represent complex local flow structures. This is achieved using a process which progressively evaluates pairs of neighbouring cells according to a distance metric that minimises error. The error is calculated from two elliptic similarity functions: one compares direction and magnitude, and the second compares location of the neighbouring vectors.

The clustering process iterates until a single cluster remains n_{root}. This is the root of a binary hierarchical tree. The clusters at each level of the tree stores the resultant vector as a product of the child pair. This results in a tree of size $n_t + n_t - 1$ where n_t is the quantity of initial clusters. The algorithm displays the clusters representative vectors at a user specified level of the hierarchical tree s_l. The accumulated error in the paired clusters is inversely proportional to s_l which is designed to directly correlate with the number of clusters selected from the hierarchical tree and therefore the number of seed locations. See subsection 2.1.3. This combined with the user defined parameters for the error function provides a flexible range of stream surface density and locality which can be tailored to the users requirements.

Step 1 initClusters()

```
1: ClusterSet s;
2: for (all cells cell_i in Ω) do
3:     c = makeCluster(cell_i);
4:     c_level = 0;
5:     add c to s;
6: end for
```

Step 2 initHashTable()

```
1: for (all clusters c_i in s) do
2:     for (all clusters c_j neighbours of c_i) do
3:         ε = clusteringError(c_i, c_j);
4:         insert pair (c_i, c_j) in increasing order of ε in a hash table;
5:     end for
6: end for
```

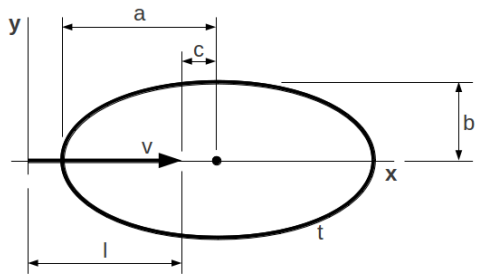
Step 3 genTree()

```
1: int l = 0;
2: for (all pairs (c_i, c_j) in increasing order in the hash table) do
3:     if (both c_i and c_j are NOTCLUSTERED) then
4:         c = mergeClusters(c_i, c_j);
5:         c_level = ++l;
6:         mark c_i and c_j as CLUSTERED;
7:         for (all neighbours n_i of c) do
8:             ε = clusteringError(c, n_i);
9:             insert pair (c, n_i) in increasing order of ε in the hash table;
10:        end for
11:    end if
12: end for return c as root of hierarchical tree;
```

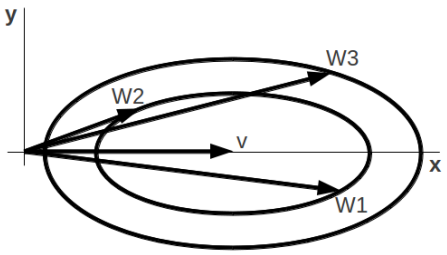
Figure 4.2: *Pseudocode of the clustering process. The code shows three main steps; first is the generation of the initial clusters, second is the pairing of the initial clusters, and third is the clustering of the pairs and new pair evaluation.*

2.1.1 Clustering Process

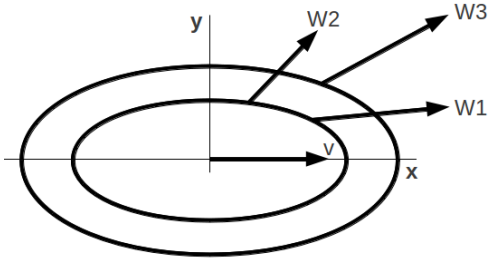
The input of the algorithm is a 3D steady state vector field $\mathbf{v}(\mathbf{p}) \in \mathbb{R}^3$ where $\mathbf{v}(\mathbf{p}) = \begin{bmatrix}\mathbf{v}_x(x,y,z) & \mathbf{v}_y(x,y,z) & \mathbf{v}_z(x,y,z)\end{bmatrix}$ for $\mathbf{p} \in \Omega$, $\mathbf{v} \in \mathbb{R}^3$ and $\Omega \subset \mathbb{R}^3$, where Ω is a 3D uniform grid. The first step requires each $(\mathbf{p}_i, \mathbf{v}(\mathbf{p}_i))$, where $\mathbf{p}_i$ is the grid location, to be stored as a cluster n_i in an initial list of clusters, or cluster set. Each n_i is evaluated against each of its neighbours in Ω. For each neighbour n_j the cluster pair (n_i, n_j) is stored in a hash table along with the derived error ε as the key. The hash table stores all (n_i, n_j) in increasing order of ε. The clustering process retrieves, in increasing order of ε from the hash table, each (n_i, n_j). If both (n_i, n_j) have not previously been clustered, then (n_i, n_j) are merged. Once merged the new cluster n_k is then compared to each of



(a) The relationship of vector **v** and the elliptical isocontour ε_τ are described in terms of parameters a, b, c, and vector length $|\mathbf{v}|$.



(b) **w1** and **w2** have equal error as they lie on the same isocontour ε_τ from the reference vector **v**. **w3** however lies on a larger isocontour.



(c) Similarly to figure 4.3(b), the location of **w1** and **w2** have equal error as they lie on the same isocontour ε_τ from the reference vector location **v**. **w3** location lies on a larger isocontour.

Figure 4.3: *An illustration of the Elliptic Similarity Function. Figure 4.3(a) describes the relationship of the parameters. Figure 4.3(b) describes the direction and magnitude components of the error function. The error function represented in figure 4.3(c) describes the relative location of a vector to a reference vector.*

its neighbours, storing each pair in the hash table in increasing order of ε. This process continues until there are no more pairs to merge. The result is a single cluster covering the entire domain with a single representative vector. The clustering process pseudocode is detailed in Figure 4.2.

2.1.2 Distance Metric and Cluster Merging

The clustering algorithm utilises two important functions. The first is the error estimation function, and the second is the merging function. The cluster error function describes the error generated by merging (n_i, n_j) i.e. the similarity between (n_i, n_j). In addition to this we implemented an alternative error function that provides additional flexibility when less similarity is desired. The cluster merging function dictates how the representative vector is calculated from (n_i, n_j).

The clustering algorithm utilises two important functions. The first is the clustering error function, and the second is the cluster merging function. The cluster error function

describes the error generated by merging (n_i, n_j) ie. the similarity between (n_i, n_j). In addition to this we implemented an alternative error function that provides additional flexibility when less similarity is desired. The cluster merging function dictates how the representative vector is calculated from the clustered pair.

The Elliptic Error Function As shown in Figure 4.3(a) a, b, and c are parameters of an elliptic contour ε_τ which represents error, or deviation from $\mathbf{v}$. The error contour is defined as:

$$f(x_\tau, y_\tau, \varepsilon_\tau) = \left[\frac{x_\tau - c(\varepsilon_\tau)}{a(\varepsilon_\tau)}\right]^2 + \left[\frac{y_\tau}{b(\varepsilon_\tau)}\right]^2 - 1 \tag{4.1}$$

The parameters a, b, and c are linear functions of ε_τ and are defined as:

$$a(\varepsilon_\tau) = C_\alpha \varepsilon_\tau \tag{4.2}$$
$$b(\varepsilon_\tau) = C_\beta \varepsilon_\tau \tag{4.3}$$
$$c(\varepsilon_\tau) = |\mathbf{v}| + C_\gamma \varepsilon_\tau \tag{4.4}$$

Solving for ε_τ when $f(x_\tau, y_\tau, \varepsilon_\tau) = 0$ and $\varepsilon_\tau > 0$, with respect to the coefficients C_α, C_β, C_γ, and vector length $|\mathbf{v}|$ yields:

$$\varepsilon_\tau = \frac{\sqrt{((x_\tau - |\mathbf{v}|)^2 + \frac{C_\alpha^2}{C_\beta^2} y_\tau^2)(C_\alpha^2 - C_\gamma^2) + C_\gamma^2 (x_\tau - |\mathbf{v}|)^2}}{C_\alpha^2 - C_\gamma^2} - \frac{C_\gamma (x_\tau - |\mathbf{v}|)}{C_\alpha^2 - C_\gamma^2} \tag{4.5}$$

Values for the coefficients C_α, C_β, and C_γ, based on the vector length $|\mathbf{v}|$ provide an aspect ratio of 1:2, and are given as:

$$C_\alpha = 2|\mathbf{v}| \tag{4.6}$$
$$C_\beta = |\mathbf{v}| \tag{4.7}$$
$$C_\gamma = |\mathbf{v}| \tag{4.8}$$

The function describes the similarity of two vectors by the quantity of error denoted by the iso error contour ε_τ. Two vectors of identical error will lie on the same isocontour eg. **w1** and **w2** in figure 4.3(b). The error of **w3** is greater than that of **w1** and **w2** relative to $\mathbf{v}$.

The relative location of $\mathbf{w}$ to $\mathbf{v}$ are similarly modelled. The vector locations described as x_ψ, y_ψ are evaluated by an elliptic function in the same way. The error is described by the isocontour is illustrated in figure 4.3(c). The error ε_ψ, when $\varepsilon_\psi(x_\psi, y_\psi) = 0$, is given as:

$$\varepsilon_\psi(x_\psi, y_\psi) = \frac{x_\psi^2}{d^2} + \frac{y_\psi^2}{e^2} - 1 \tag{4.9}$$

The user can specify the the aspect ratio $\eta_\psi = d/e$ of the elliptic iso contour where $\eta_\psi[0, \ldots, 1]$, $d + e = 1$, and $\eta_\psi = 0.5$ for a ratio of 50:50 (1/1). A ratio of 75:25 (3/1)

($\eta_\psi = 0.75$) will emphasis clustering along the direction of the reference vector, and a ratio of 25:75 (1/3) ($\eta_\psi = 0.25$) will emphasis clustering adjacent to the direction of the reference vector. The linear combination of the error functions ε_τ, which represents the error between vectors directions and magnitudes, and ε_ψ, which represents relative location error, is combined as:

$$\varepsilon = \eta_{\tau\psi}\varepsilon_\psi + (1 - \eta_{\tau\psi})\varepsilon_\tau \tag{4.10}$$

where $\eta_{\tau\psi}[0,\ldots,1]$ is a user controlled coefficient which emphasises either ε_τ or ε_ψ in the clustering process.

Experiments with Elliptic Error Functions Our experiments with the elliptic error functions produced interesting results when used for stream surface seeding. However we observe limitations which are addressed with a customised distance function.

When modifying the input parameters, a range of characteristics are observed. For example, when $\eta_{\tau\psi} = 0.9$ emphasising position, and $\eta_\psi = 0.75$ emphasising neighbours along the vector direction, the process produces an evenly distributed set of clusters at a given s_l. Setting $\eta_{\tau\psi} = 0.9$ emphasising position and $\eta_\psi = 0.25$ emphasising neighbours orthogonal to the vector direction, concentrates the clusters around areas of curved flow.

Our aim is to generate seeding curves in locations adjacent to the flow structures with minimal density i.e. enough to represent the flow structure without clutter. Seeding curves at the centre of highly curved structures such as vortices is not ideal and may result in complex surfaces that are difficult to discern. The complexities include high curvature, shearing, crossing, and diverging surfaces. The ideal solution is to construct the minimum quantity of seeding curves at locations of interesting flow and at a non zero distance from the centres of complex, curved structures. Flow structures are characterised by their curvature and velocity gradient. The vortex is one example of this. See Figure 4.4.

Generating clusters to emphasise pairing in areas where there is a non zero velocity gradient motivates us to define a parameter which supports this. For example if the error function returned lower ε for cluster pairs which have a greater differences in magnitude than their neighbours, while favouring orthogonal clusters e.g. $\eta_\psi = 0.25$, we can predict a concentration of clusters in areas of greater velocity gradient. Extending this strategy to vector direction, we can specify a concentration of clustering in areas of high curvature while emphasising clusters orthogonal to the vector direction e.g. $\eta_\psi = 0.25$. In the remainder of this section we present our customised distance function which is designed to implicitly capture, in combination, areas of non zero velocity gradient and areas of curved or rotating flow.

A Customised Distance Function To address the hypothesis in the previous subsection, we are motivated to create a customised distance function. The distance function will be required to effectively pattern match attributes of the compared vectors using

unsupervised machine learning, or agglomerative hierarchical (binary) clustering. To achieve this our parameters are constructed to return low error not just for similar, but also dissimilar directions or magnitudes of a quantified amount.

To achieve this, we start by decoupling the direction $\hat{\mathbf{v}}$ and magnitude $|\mathbf{v}|$ components of the original distance function. This allows the user to treat direction and magnitude independently providing greater control over the clustering process. This enables emphasising a particular direction or difference in magnitude rather than pure similarity. This is effectively pattern matching, for example we can now return low error values when vectors are orthogonal to each other rather than just parallel.

To incorporate the direction ε_δ, magnitude ε_μ, and position ε_ψ functions, we simply linearly combine them. Combining the functions in this way allows the user to more easily comprehend the effect of changing those parameters. The user options for specifying the emphasis of a particular direction $\eta_\delta \in [0,1]$, difference in magnitude $\eta_\mu \in [0,1]$ or position $\eta_\psi \in [0,1]$ are included:

$$\varepsilon = \varepsilon_\delta(\eta_\delta) + \varepsilon_\mu(\eta_\mu) + \varepsilon_\psi(\eta_\psi) \tag{4.11}$$

where $\varepsilon \in [0,1]$.

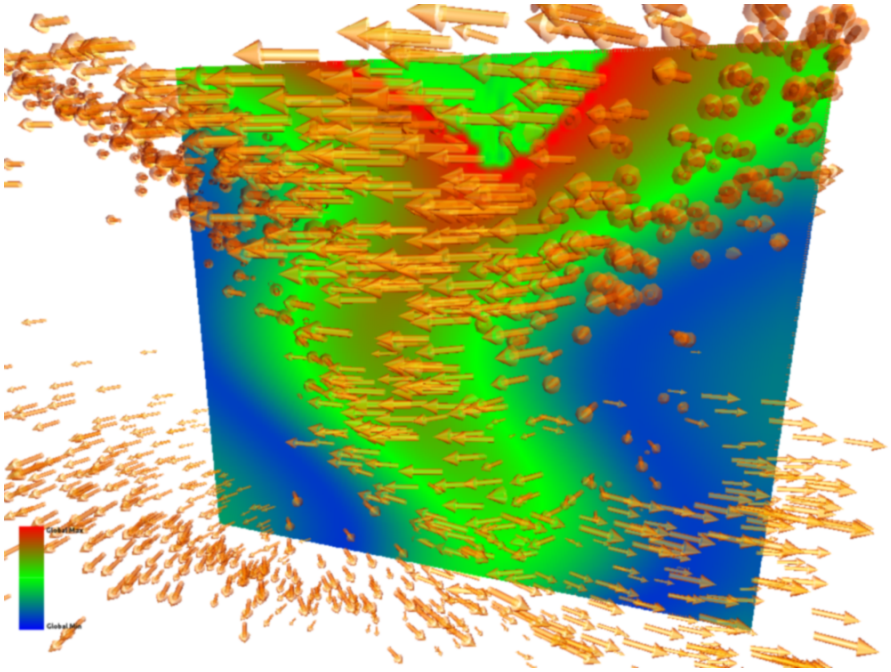


Figure 4.4: *A 2D slice taken at the centre of the Tornado simulation. Velocity is mapped to colour where blue is minimum velocity and red is maximum. This illustrates the velocity gradient initially increasing and then decreasing away from the centre of the vortex.*

Direction The user controlled coefficient η_δ corresponds to parallelism or orthogonality. Setting $\eta_\delta = 0$ emphasises parallel vectors $\mathbf{v} \parallel \mathbf{w}$, whereas $\eta_\delta = 1$ emphasises opposing vectors. $\eta_\delta = 0.5$ emphasises orthogonal vectors $\mathbf{v} \perp \mathbf{w}$. The linear combination is specified as:

$$\varepsilon_\delta(\eta_\delta) = (\eta_\delta \cdot \delta) + (1 - \eta_\delta) \cdot (1 - \delta) \tag{4.12}$$

Where $\varepsilon_\delta \in [0,1]$ represents the direction centred error, with lower values favoured by the clustering process, and $\delta \in [0,1]$ is:

$$\delta = \frac{(\hat{\mathbf{w}} \cdot \hat{\mathbf{v}}) + 1}{2} \tag{4.13}$$

Magnitude Setting $\eta_\mu = 0$ emphasises vectors of equal length $|\mathbf{v}| = |\mathbf{w}|$, whereas $\eta_\mu = 1$ emphasises vectors of different length $|\mathbf{v}| \neq |\mathbf{w}|$. The linear combination is:

$$\varepsilon_\mu(\eta_\mu) = (\eta_\mu \cdot \mu) + (1 - \eta_\mu) \cdot (1 - \mu) \tag{4.14}$$

Where $\varepsilon_\mu \in [0,1]$ represents the magnitude centred error, with lower values favoured by the clustering process, and $\mu \in [0,1]$ is the absolute value of μ' defined as:

$$\mu' = |\mathbf{v}|/(|\mathbf{v}| + |\mathbf{w}|) - |\mathbf{w}|/(|\mathbf{v}| + |\mathbf{w}|) \tag{4.15}$$

Position For position we use the elliptic error function by Telea and Van Wijk [TvW99]. For consistency our user controlled coefficient η_ψ corresponds to B, and $\varepsilon_\psi(\eta_\psi)$ corresponds to s where $\varepsilon_\psi \in [0,1]$.

Default Settings The process of arriving at a set of default values for the user is based on visual insight gained from the visualisations rather than formal mathematical analysis. This approach was undertaken with the aid of domain experts. As a result of feedback from experimenting with the customised distance function parameters, we derived two sets of values which provide the user with either feature centred or overview settings. For the feature centred settings $\eta_\psi = 0.25$, $\eta_\delta = 0.3$, $\eta_\mu = 0.2$. For the overview settings $\eta_\psi = 0.75$, $\eta_\delta = 0.0$, $\eta_\mu = 0.0$. From hereafter we will refer to these settings as feature centred or overview. Further discussions of clustering parameters reside in Section 3.

The Merging Function The union of a pair of clusters (n_i, n_j) results in a parent cluster n_k. The new n_k stores a representative vector $\mathbf{v}(n)$ and location $\mathbf{p}(n)$. This is achieved by computing a volume weighted average of (n_i, n_j) representative vectors and locations. Each level of the tree contains clusters of increasing ε.

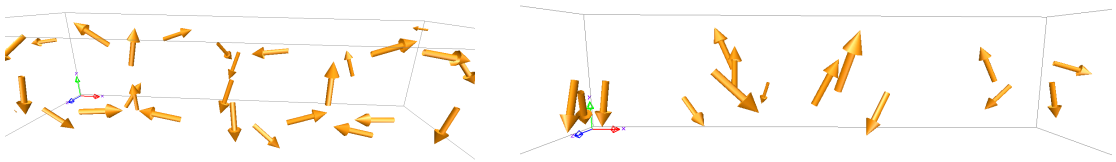


Figure 4.5: *The left image is the 3D Bernard flow simulation visualised with* $\eta_\psi = 0.25$, $\eta_\delta = 0.0$ $\eta_\mu = 0.5$ *and* $s_l = 27$. *The glyphs demonstrate clustering in the vicinity of the velocity gradient associated with the vortices. The right image demonstrates cluster representations focusing around cores of the vortices. This is achieved using parameters which emphasise clustering of vectors with orthogonal directions.* $s_l = 27$, $\eta_\psi = 0.5$, $\eta_\delta = 0.5$ *and* $\eta_\mu = 0.0$.

2.1.3 Simplification Level

The simplification level s_l is an option for the user to select clusters or seeding locations from the cluster hierarchy at rendering time. Error metric ε is used in the first step of the clustering algorithm to place pairs (n_i, n_j) in the hash table in order of increasing ε. A second step clusters them in this order marking each new cluster with a unique order number l incremented from 0. As a result ε is directly related to l. Once the tree has been built this order number l is compared to the user parameter s_l to determine cluster selection for seeding or display. The following describes the computational relationship of s_l, l and ε and how the clusters are selected from the tree. Because we use a binary tree the root cluster's level $l(n_{root}) = n_t - 1$ where n_t is the quantity of initial clusters.

To select clusters from the tree, the user specifies s_l in the range $s_l \in [1, n_t - 1]$. We next calculate l where $l = n_t - s_l$. The level l is then used to search the tree for all clusters with levels $l(n) \leq l$ with parents having $l(n) > l$. The set of clusters which meet this condition are returned as the seeding locations. s_l is designed to be equal to the number of clusters or seeding locations, and is inversely proportional to ε. s_l provides consistency for the user across different simulation data for selecting the quantity of seed locations. See Figure 4.5. However, the choice of cluster quantity is partially craft as the user will require foresight of the feature quantity.

2.2 Curvature Field Derivation

From the vector field $\mathbf{v}(\mathbf{p})$ we derive the curvature field Ω_c. This step allows the curvature data to be sampled or interpolated on demand. The role of Ω_c is to support seeding curve computation (Section 2.3) and illustrative techniques for rendering the stream surfaces (Section 2.5). Ω_c is not needed for the clustering stage of the pipeline.

Each curvature sample $\mathbf{c}(\mathbf{p}) \in \Omega_c$ defined for $\mathbf{p} \in \mathbb{R}^3$ and $\Omega_c \subset \mathbb{R}^3$, is derived by applying a combination of operators to the vector field. Ω_c is derived from the first derivative $\mathbf{v}$ and second derivative $\mathbf{a}$ of the flow field. The second derivative $\mathbf{a}$ is accel-

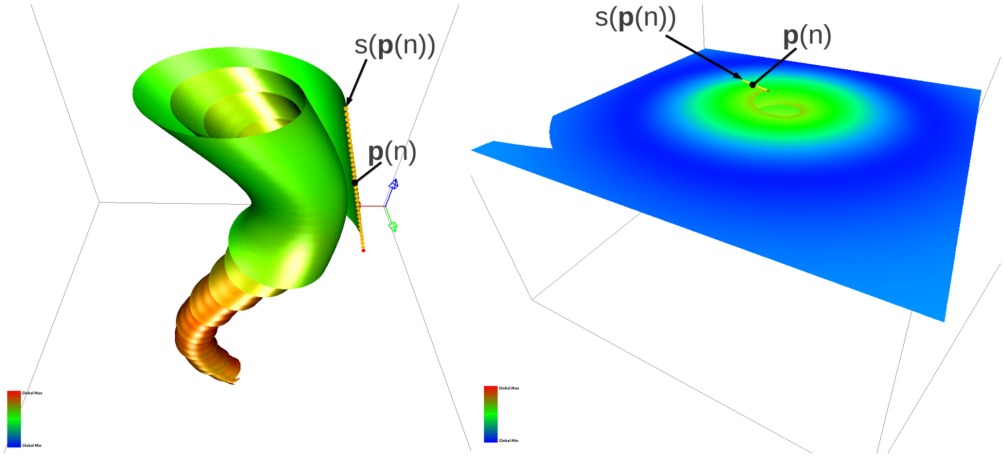


Figure 4.6: *Rotating flow is more intuitively represented in the left figure. The left figure uses a surface tangent to its curvature and parallel with the axis of rotation. The right figure is represented by a surface perpendicular to the axis of rotation and flow curvature.*

eration and defined as $\mathbf{a} = (\nabla \mathbf{v})\mathbf{v}$ where $\nabla \mathbf{v}$ is the Jacobian of the velocity field. Steady state curvature [Rot00] is defined as:

$$\mathbf{c} = \frac{\mathbf{v} \times \mathbf{a}}{|\mathbf{v}|^3} \tag{4.16}$$

2.3 Seeding Curve Computation

The seeding curve generation commences following the clustering. Every n_i in the selected representation stores information regarding its location $\mathbf{p}(n)$, its level $l(n)$, its error $\varepsilon(n)$, its representative vector $\mathbf{v}(n)$ and spatial volume $vol(n)$. $\mathbf{p}(n)$ defines the centre of each new seeding curve $s(\mathbf{p}(n))$. Location alone is not enough to generate effective seeding curves. The locations given by the cluster representation provide interesting seeding positions that capture the local characteristics of the flow. To take advantage of the potential seeding locations we ensure the seeding curves are positioned, orientated and scaled according to the local flow characteristics. The position is given by the cluster. The orientation and size of the seeding curves require analysis.

2.3.1 Seeding Curve Orientation

The orientation of the seeding curve stems from the observation that seeding curves generally produce informative surfaces maximising coverage when orthogonal to the flow. In addition the seeding curve should be oriented with respect to the local flow structures. If this point is ignored the surface can impart a less useful visualisation to the viewer. For example if a surface is tangent to the local curvature and parallel to

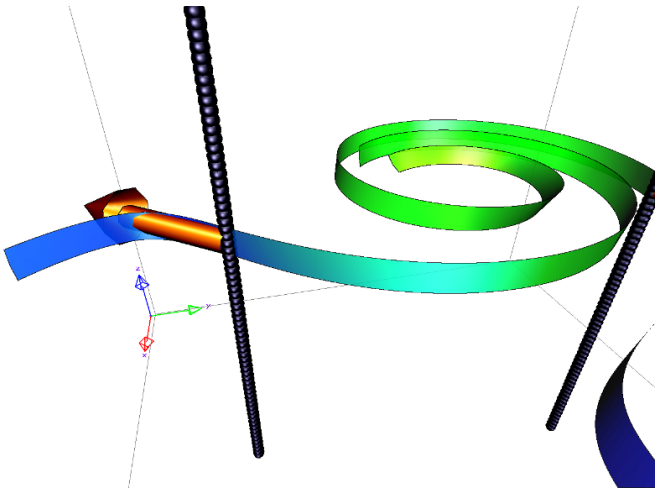


Figure 4.7: *The Tornado simulation illustrates the alignment of seeding curves to the curvature field. The feature centred values are specified and $s_l = 2$. The arrow glyph visualises the representative vector and location. The seeding curves (black) orientation to flow curvature is emphasised using stream ribbons.*

the axis of rotation it intuitively represents rotating flow. If the flow is represented by a surface perpendicular to the axis of rotation and flow curvature, the representation is less informative. See Figure 4.6. We position a straight seeding curve $s(\mathbf{p}(n))$, centred at $\mathbf{p}(n)$, orientated in line with the local curvature vector $\mathbf{c}(\mathbf{p}(n))$. The curvature vector $\mathbf{c}(\mathbf{p}(n))$ is sampled from the curvature field Ω_c at $\mathbf{p}(n)$.

The result of this approach is a seeding curve which is orthogonal to the local flow at its centre. The seeding curve is also orientated such that it is parallel to the axis of local flow rotation. In Figure 4.7 we see examples of the proposed seeding curve orientation. The Figure shows how the seeding curve orientation relates to the flow characteristics, and the potential to provide intuitive visualisations of the nearby flow structures.

2.3.2 Seeding Curve Length

The length of a seeding curve $s(\mathbf{p}(n_i))$ is proportional to the volume of cluster n_i. The volume of cluster n_i is its spatial volume. Specifying the length of the seeding curve as equivalent to the clusters volume may lead to the curve exiting the boundaries of Ω. To refine this, we use the cube root of the cluster volume:

$$length(s) = \sqrt[3]{vol(n_i)} \tag{4.17}$$

This is based on the premise that a single cluster representing the whole domain would require a seeding curve which extends to its boundaries. This approach works well for all the data we tested. However, the user is also provided with an optional scaling coefficient for further customisation. This enables shorter seeding curves when seeding

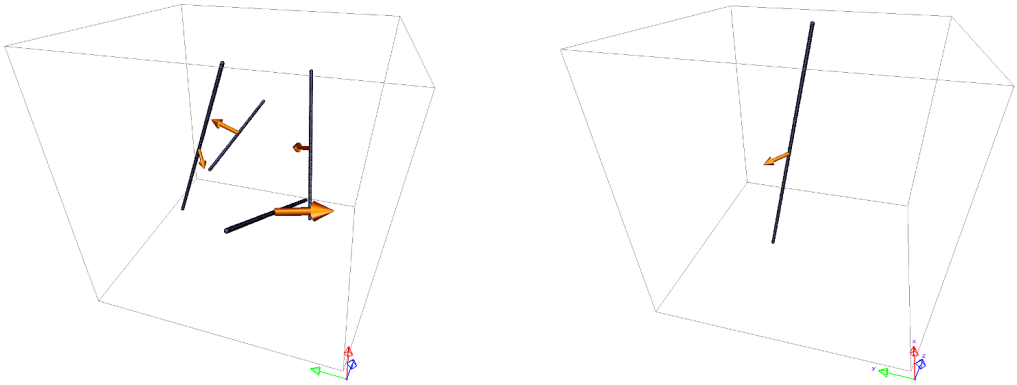


Figure 4.8: *These images show that the seeding curve length is proportional to the volume weighted average of the clusters children. The left image is represented by 4 clusters and the right image by 1.*

with a denser cluster representations, and vice versa. Figure 4.8 illustrates different seeding curve lengths proportional to cluster density.

2.3.3 Seeding Curve Integration

Our experiments with this method indicated that further refinement of the seeding curve is desired. In areas of flow which demonstrate high curvature seeding curves may cross each other. See Figure 4.9. Longer seeding curves can present problems. As the distance away from the seeding curve centre increases, its orientation relative to the local flow direction may change. For example the seeding curve is orthogonal to the flow at its centre, but may be in line with the flow at its extremities. Additionally the seeding curves do not follow the local flow structures that we aim to capture with the clustering process. For example a straight seeding curve generated neighbouring a vortex curved along its core line would either extend away from or cross the vortex.

To address these challenges we further enhance the seeding curve generation. A logical approach is to exploit the curvature field Ω_c. The curvature vector is the cross product of velocity and acceleration, and therefore always orthogonal to velocity. This

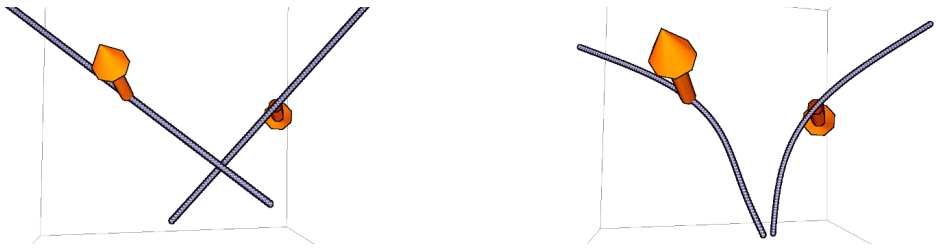


Figure 4.9: *These illustrations demonstrate straight seeding curves vs integrated seeding curves. The representative vectors and locations are visualised with arrow glyphs. The left image shows the seeding curves crossing each other when aligned with the curvature vector. The right image shows the seeding curves following the curvature, not crossing, and remain orthogonal to the flow.*

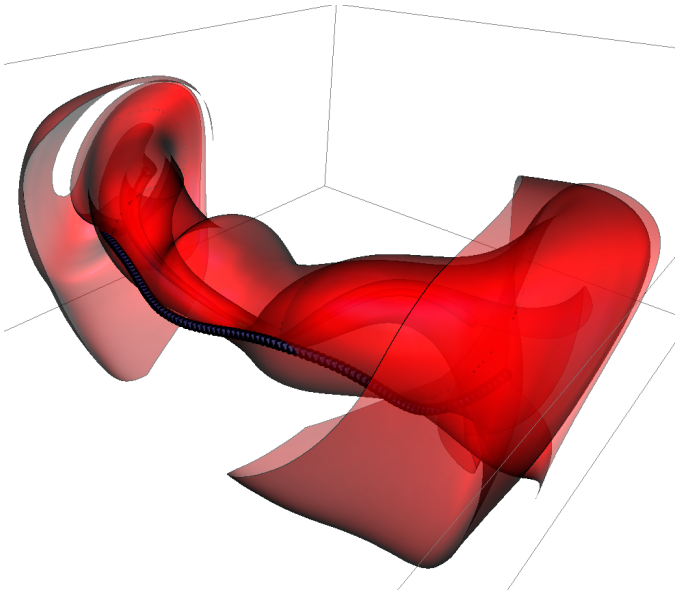


Figure 4.10: *This illustration of the Bernard simulation demonstrates the seeding curve following the flow structure. The seeding location derived from the clustering is at the centre of the curve (black).*

is an ideal candidate as we generate a seeding curve orthogonal to the flow along its the complete length while following the local curvature of the flow field.

Starting at $\mathbf{p}(n)$ the seeding curve $s(\Omega_c(\mathbf{p}(n)))$ is generated by integrating through the curvature field $\Omega_c(\mathbf{p}(n))$. The integration step is defined by the required seeding curve discretisation. See section 2.4. The length of the curve is constrained as previously described. The integration is achieved using a fourth order Runge-Kutta integrator, sampling the curvature field $\Omega_c(\mathbf{p}(n))$, in forward and backward directions. Examples of this method can be seen in Figure 4.10. It is possible within the curvature field to find degenerate points if $\mathbf{v}$ and $\mathbf{a}$ are equal. We handle the degenerate points by simply terminating the integration.

2.4 Stream Surface Generation

Our work utilises a standard solution to stream surface computation (See [GKT*08] using the integration scheme in Appendix A). We allow the user to select downstream or/and upstream stream propagation e.g. forward and reverse integration. By default surfaces are terminated when they leave the domain, enter a periodic orbit, or reach a predetermined maximum length. The user has an option to control the length.

During the construction of the stream surface it is useful to sample attributes of the flow which can be stored with the vertex data representing the surface. This data is then be passed to the rendering pipeline for further processing as described in 2.5. Attributes of use can include colour mapping $|\mathbf{v}|$. The alpha channel can be mapped to $|\mathbf{c}|$. This is particularly useful when rendering semi transparent surfaces with depth peeling as described in 2.5.

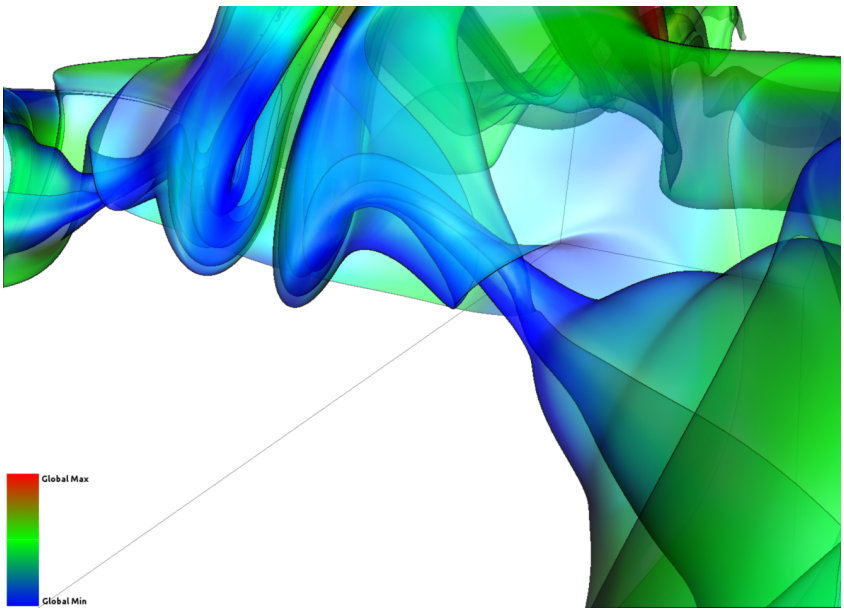


Figure 4.11: *The flow past a cuboid simulation demonstrating illustrative techniques on a stream surface. The surface is generated neighbouring a double vortex structure emanating from a critical point.*

2.5 Rendering Enhancements

A number of techniques are implemented to aid the viewer in perception of the resulting visualisation. The techniques include the use of transparency, colour, silhouette edge highlighting, lighting and shading. We incorporate semi transparency with a combined approach. First as part of the stream surface construction algorithm we can assign alpha values from Ω_c. For example with higher curvature $|\mathbf{c}|$ we assign a higher alpha value α_c. Note: $max(|\mathbf{c}|)$ is the maximum curvature in Ω_c, and is used to normalise $\alpha_c \in [0,1]$. This has the effect of increasing opacity of the inner structures of curved surfaces:

$$\alpha_c = \frac{|\mathbf{c}|}{max(|\mathbf{c}|)} \tag{4.18}$$

The second part of our combined approach is a view dependent strategy which uses the relationship α_v between the view normal $\hat{\mathbf{n}}_\mathbf{v}$ and the surface normal $\hat{\mathbf{n}}_\mathbf{s}$ defined as:

$$\alpha_v = \frac{2cos^{-1}(\hat{\mathbf{n}}_\mathbf{s} \cdot \hat{\mathbf{n}}_\mathbf{v})}{\pi} \tag{4.19}$$

where $cos^{-1}(\hat{\mathbf{n}}_\mathbf{s} \cdot \hat{\mathbf{n}}_\mathbf{v})$ is given in radians. This angle-based relationship has the effect of increasing the transparency when the surface tends to face the view port, or reducing transparency when the surface is orthogonal to the view port. A linear relationship with a user specified bias η_α is used to combine these aspects:

$$\alpha = \eta_\alpha \alpha_c + (1 - \eta_\alpha)\alpha_v \tag{4.20}$$

The application of transparency to visualisations poses problems relating to the order of primitive rendering. To solve this issue we use depth peeling, an order independent transparency technique presented by Bavoli and Myers [BM08].

Silhouette edge highlighting is used to help the viewer in perceiving where the surfaces curve away from the viewer, and enhance surface edges. Silhouette highlighting utilises a simple Gaussian kernel in image space [MH02]. The technique is applied to a depth image from the current render pass and is blended to the frame buffer. This way it can be used during normal scene rendering or interleaved with the depth peeling loop.

In addition, reducing the saturation of colour as the surfaces curve away from the viewer further enhances the perception of shape. To implement this we modify the magnitude of the *RGB* values during the rendering process. An efficient method is to reuse α_v. This angle-based relationship is then used to scale the *RGB* values RGB_{in} sent to the pipeline: $RGB = RGB_{in}(1 - \eta_{RGB}\alpha_v)$ where η_{RGB} is a user supplied bias. A suitable value for η_{RGB} and η_α derived from experimentation is 0.5. In some cases the results of this technique can be misleading. The change in transparency or saturation can influence some aspects of the visualisation. For example, it can alter the perception of the base colour, misdirecting the observer as to the true value in the region of interest. Thus further refinement, including toggling these techniques on and off, is made available interactively to users if they so desire. Examples of these settings are shown in Figure 4.11.

3 Results

The key to an informative visualisation is the ability to capture the flow characteristics and to represent different data attributes. In this section we discuss the results and the performance of our algorithm. The results demonstrate the algorithm applied to a range of data, both analytical and computational.

The first two datasets highlighted are the Tornado and Cuboid data. These two datasets are used in Chapter 3, and used here for reference in addition to the following analysis. The Bernard numerical simulation offers the opportunity to test our algorithm on a vector field with multiple vortex structures located across the domain. Hurricane Isabel is a large dataset demonstrating flow characteristics related to the Tornado data. This dataset is used to challenge the computational speed and memory usage of our algorithm. The different simulations provide a diverse environment testing the robustness of the algorithm. We demonstrate the clustering, seeding, and illustration options on each of the simulations. The datasets studied in this section are all normalised velocity fields in the range [0,1].

Hurricane Isabel This is a simulation from the IEEE Visualisation Contest 2004. The hurricane modelled is Hurricane Isabel which occurred in September of 2003. The clustering and surface seeding process is applied to the Isabel flow data with different sets of parameters designed to show the flexibility of the algorithm.

The first set of parameters are $\eta_\psi = 0.25$, $\eta_\delta = 0.3$, $\eta_\mu = 0.2$ (feature centred), with $s_l = 3$. This produces clusters in the region of the hurricane vortex. Figure 4.12 shows the stream surfaces which capture the vortex. The visualisation uses transparency and edge highlighting to enhance the perception of the rotating flow, and capture the inner structure of the vortex. The use of a higher s_l value produces more seeding locations in this area, moving outwards as the quantity increases. This may be useful in some cases but is likely to become cluttered. The second set of parameters are designed to give a more evenly distributed set of seeding locations. The parameters are $\eta_\psi = 0.75$, $\eta_\delta = 0.0$, $\eta_\mu = 0.0$ (overview), with $s_l = 30$.

In Figure 4.13 an overview of the simulation can be seen. The illustrative techniques support the perception of the flow characteristics with transparency enabling the user to understand the inner flow structure of the vortex. It can be seen from this visualisation how the general flow behaviour interacts with the eye of the hurricane. A second vortex like structure can be seen in the mid left of Figure 4.13, located above the coast.

Flow Past A Cuboid This simulation demonstrates flow past a Cuboid over 102 time steps. For our experiments we use the last time step containing fully formed flow structures. The clustering process is applied with one set of parameters at different simplifi-

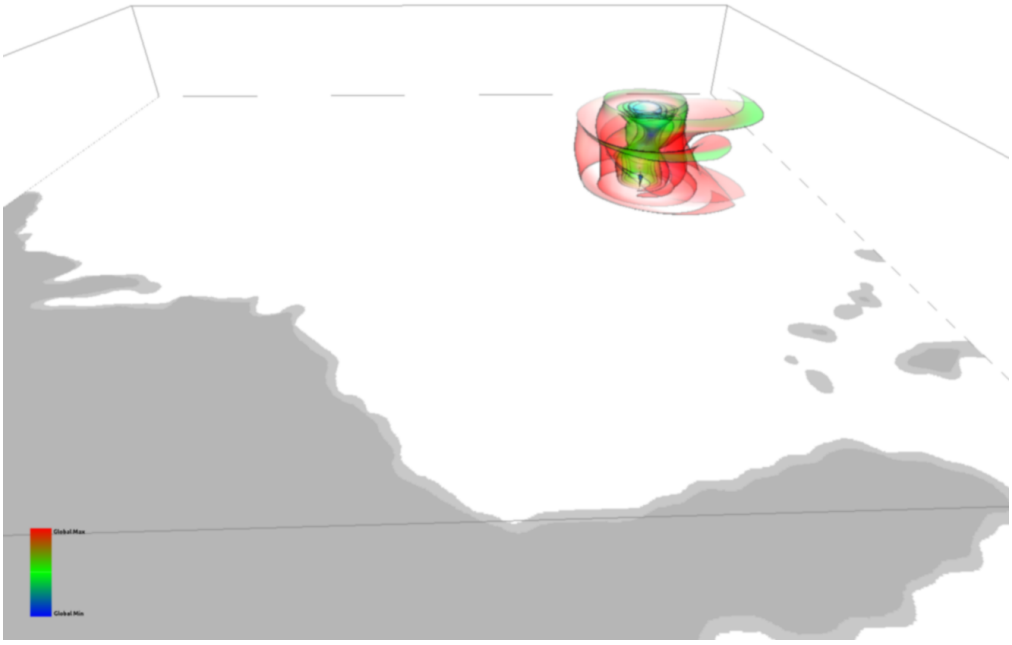


Figure 4.12: *Hurricane Isabel data visualised with automatic stream surfaces. Colour is mapped to velocity, and opacity is mapped to vector field curvature. This visualisation emphasises the eye of the hurricane captured by stream surfaces rendered with edge highlighting and view dependent colour saturation. The inner structure of the vortex tends away from blue as the velocity increases away from the centre to the vortex.*

cation levels. We use $\eta_\psi = 0.25$, $\eta_\delta = 0.3$, $\eta_\mu = 0.2$ (feature centred), with $s_l = 2$, and $s_l = 4$ respectively.

Figure 4.14 demonstrates the seeding algorithm at $s_l = 2$. The double vortex structure is clearly visible. Figure 4.15 illustrates the ability of the user to change the level of simplification to $s_l = 4$, thus generating a denser visualisation by combining the two sets of surfaces providing better contextual information. The visualisation captures the prominent characteristics of the vortex shedding behind the cuboid.

Bernard Flow The Bernard flow data is a numerical simulation defined on a regular grid [WSE05]. The simulation demonstrates thermal motion as a result of convection. The analysts are interested in the interaction of the thermal currents, and the structure of the interacting vortices. The clustering process is applied to the Bernard flow data with $\eta_\psi = 0.25$, $\eta_\delta = 0.3$, $\eta_\mu = 0.2$ (feature centred). Once the clustering is complete the user now has the option to specify the level of simplification s_l. A slider is used to adjust the level of detail with visual feedback in the form of glyphs rendered at the cluster locations. For this visualisation we specify $s_l = 4$.

Figure 4.16 shows the final surfaces rendered with transparency. The coefficient $\eta_\alpha = 0.5$ for this visualisation demonstrates the usefulness of opacity mapped to Ω_c. Edge highlighting is used to emphasise the surface boundaries with the surface colour darkened as the surface curves away from the view port. The curved surfaces and vortex cores are clearly emphasised using these techniques. This visualisation is further en-

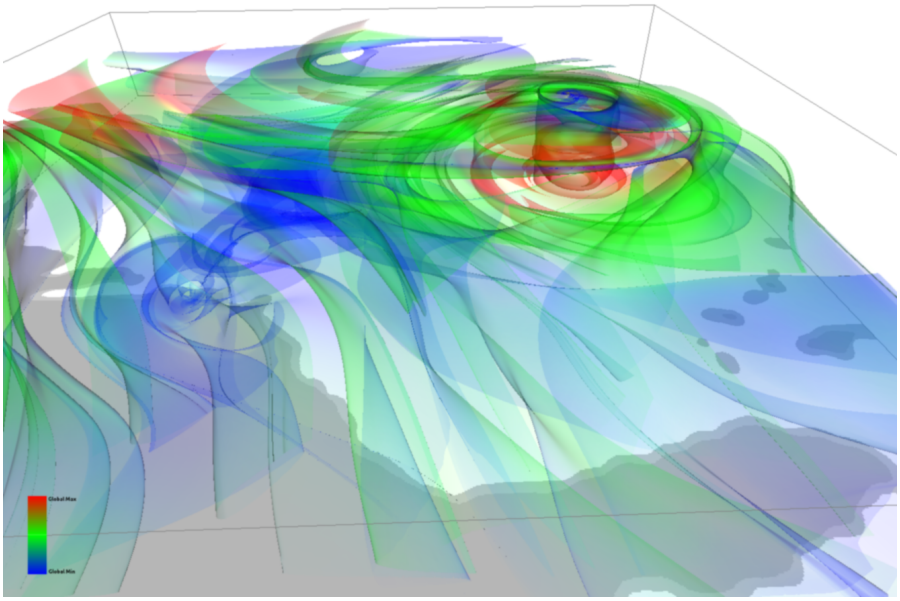


Figure 4.13: *Hurricane Isabel visualised using settings designed to give a broader representation. A second smaller vortex like structure can be seen on the coast in the mid left of this still image. Stream surfaces are rendered with edge highlighting and view dependent colour saturation. The inner structures of the simulation are viewed with additional transparency.*

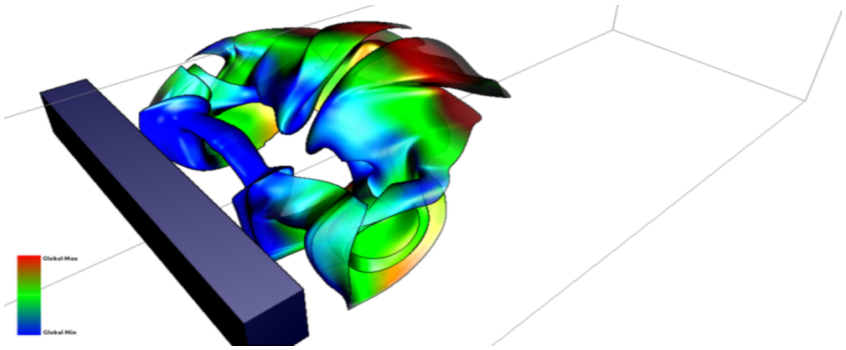


Figure 4.14: *This visualisation of the flow past a cuboid is performed with $s_l = 2$. The seeding curve follows a neighbouring double vortex structure emanating from a saddle point. The stream surface is integrated down stream, and is rendered using illustrative techniques.*

hanced with the use of colour mapped to $|\mathbf{v}|$. The unit vectors in this numerical data give a constant colour across the surfaces, except in areas of degenerate velocity. This is useful for highlighting critical points across the domain. Figure 4.17 highlights these saddle points at the centre and ends of the four double vortices.

Tornado The Tornado data is an analytical simulation defined on a regular grid. The data domain is 64 x 64 x 64 and simulates the flow structure of a fully formed tornado. The clustering process is applied to this data with one set of parameters at different representation levels. We use $\varepsilon_\psi = 0.25$, $\varepsilon_\delta = 0.3$, $\varepsilon_\mu = 0.2$, with an s_l of 5. Figure 4.18 illustrates the effectiveness of the algorithm in representing the flow characteristics. The Tornado funnel is captured by the stream surfaces while relating the flow away from the funnel to the circulating flow within the vortex core. This visualisation further demonstrates the focus and context nature of this algorithm.

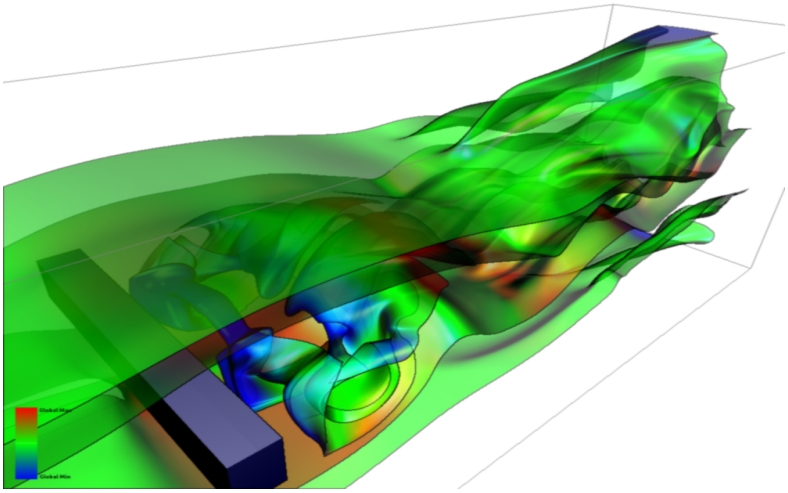


Figure 4.15: *This image highlights the ability of our algorithm to provide both overview and feature centred within the same visualisation. The vortex shedding is represented by the stream surfaces along with its relation to the rest of the flow field.*

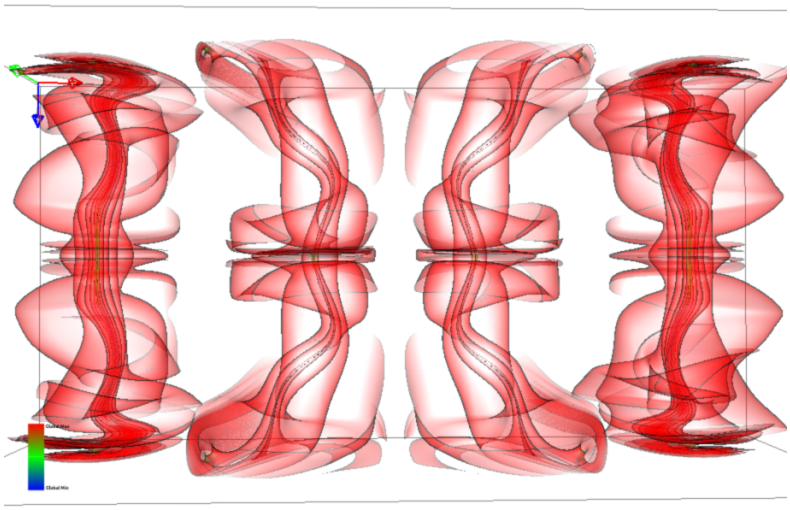


Figure 4.16: *The Bernard Flow numerical simulation visualised using our algorithm. The seeding locations are derived from the clustering process using the feature centred parameters and $s_l = 4$. The four main vortex structures are clearly emphasised. The thermal motion of the flow field is captured with our framework. The figure shows the surfaces rendered with transparency mapped to α_c and α_v.*

Parameters While the selection of a simplification level is simple, as it directly correlates to the number of seeding locations, its selection is important as the final visualisation can easily become over populated or too sparse. The selection of parameters is important for producing the type of visualisation required by the user as they are, although progressive, sensitive to change. The two proposed default settings are demonstrated in this section. It was found from experimentation that careful thought and a general understanding of fluid flow is required to design sets of parameters that are useful to the flow engineer. A poor choice of parameters could lead to cluttered visualisations of little meaning to an engineer, similar to a naive approach (See Figure 4.19). These ideas are further discussed in Section 4.

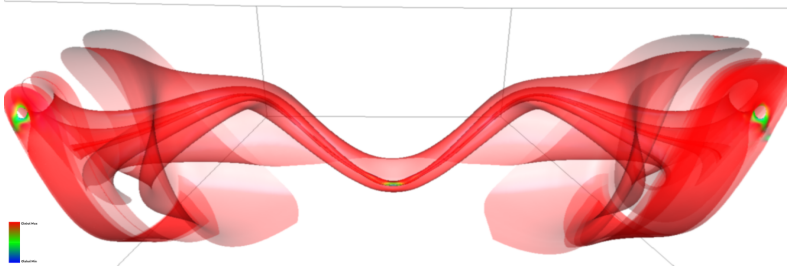


Figure 4.17: *The Bernard Flow numerical simulation visualised with $s_l = 4$, and the feature centred parameters. This figure shows one of the 4 surfaces which highlights saddle points within the domain. The stream surface is rendered red, while the degenerate areas are highlighted green to blue.*

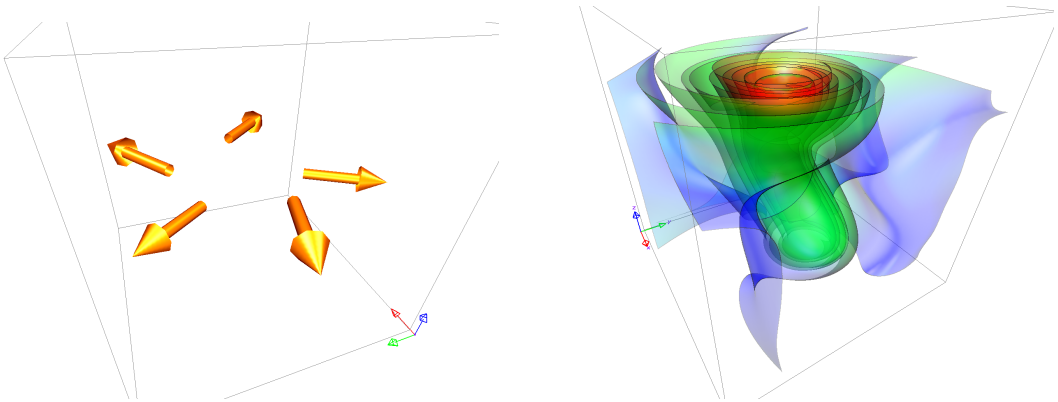


Figure 4.18: *The Tornado simulation visualised using our algorithm. The seeding locations are derived from the clustering process using the feature centred parameters and $s_l = 5$. The left image shows the cluster centres represented by the base of the arrow glyphs. The arrow glyphs represent the cumulative velocity of the cluster. The right image shows the surfaces rendered with transparency mapped to α_c and α_v.*

Clustering Performance		
Data	No. of Clusters	Time [ms]
Hurricane Isabel	49,999,999	542,452
Bernard Flow	524,287	2,908
Tornado	524,287	2,908
Cuboid	1,179,647	6,898
Lorenz Attractor	4,194,303	34,812

Table 4.1: *Clustering performance of a range of simulations.*

Performance Our algorithm is implemented in C++ and QT4 on a PC with an NVIDIA GeForce GTX480, an Intel quad core 2.8GHz CPU with 8GB RAM. The bottleneck in the performance of our algorithm is the clustering step as seen in Table 4.1. The clustering process compares closely with previous vector field clustering algorithms that operate on uniform grids [PGL*12].

The field derivation ranges from 8ms for 262,144 data samples to 958ms for 25,000,000 data samples. The seeding curve generation is a function of s_l and curve length which decreases as s_l increases. Timings are from 15ms for $s_l = 30$ to 35ms for $s_l = 100$. Stream surface rendering performance is a function of surface size and complexity e.g. the number of vertices and depth peeling layers. We achieve 20fps+ when rendering meshes of 400k vertices while utilising 10 depth layers. This is comparable to the work by Born et al. [BWF*10] and Hummel et al. [HGH*10] who discuss surface rendering performance in detail.

4 Domain Expert Feedback

To study the usefulness of the proposed framework in the visualisation of complex CFD data, we have shown our visualisation results and system to a number of CFD experts, i.e., Dr. Malki, Dr. Masters, and Dr. Croft, co authors of this chapter. This section summarises their positive and constructive feedback.

One of the most commonly used software packages for the visualisation of numerical simulation results is Tecplot, which does not offer the user the option to create stream surfaces (only stream tubes). An example is a vertical axis wind turbine[MMWC11], where the classical analysis partitions the flow into vertical slices. Both streamlines and stream surfaces have their uses, however, where 2D images are extracted for use in reports or presentations and the user cannot rotate the model to evaluate it from different viewing angles, streamlines can be quite messy and difficult to comprehend. Stream surfaces on the other hand are much neater and clearer showing how different layers of flow may fold over each other.

Surfaces are very helpful in defining boundaries in the flow. The particular approach of this chapter, where the surface seeding line is curved in response to the physical data, gives a surface that is analogous to the classical definition of the boundary of a stream tube [DGS79] or stream slice [Par82] [Hom91]. This gives the surface two properties that are very useful. First, it is a solid surface partitioning the flow into regions. Second, the surface relates (approximately) to the boundaries of stream tubes, when you realise that a stream tube can be any arbitrary shape [Hom91].

Compared to manual seeding there is an advantage of using automated methods. They are able to identify regions where interesting features occur within the flow, which may otherwise be overlooked. This is particularly relevant where the user does not have a clear understanding of the flow structure through the domain. The user may not necessarily be able to identify the locations of such features.

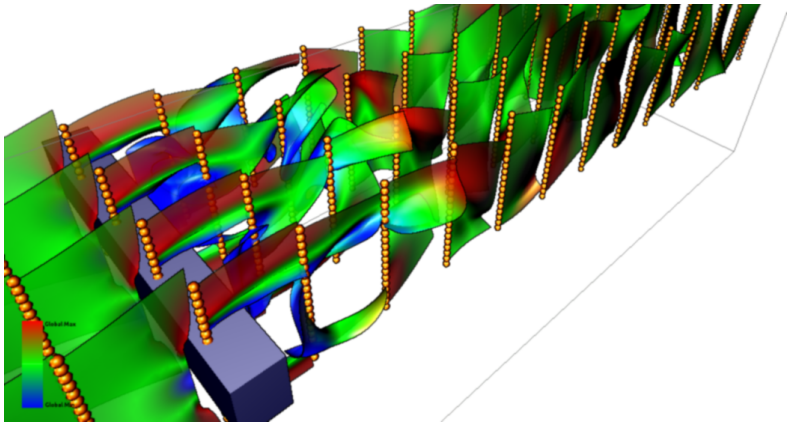


Figure 4.19: *A naive seeding approach to the flow past a cuboid simulation. Although the range of illustrative techniques are applied the perception of the flow characteristics is limited. The surfaces are seeded at regular intervals at a consistent orientation across the domain.*

The experts from the CFD community welcome the flexibility of control of seeding parameters, although it may be sufficient to provide the user with a few standard parameter sets to quickly provide the user with alternative views of the data. However, there are likely to be situations where the user is particularly interested in a specific region, even if it is not necessarily the most exciting location in terms of flow structure. It would be useful in such situations for the user to have more control over what is shown, in what level of detail and where within the domain.

Varying the number of clusters enables the user to evaluate the flow structure in different levels of detail. Fewer clusters can be used to evaluate the most complex regions of the flow, such as the swirling region of a turbine wake [MMWC11], whilst the number of clusters can be significantly increased to evaluate the far field regions and obtain a more general picture of the flow throughout the entire domain. Together, these can be used to build a more complete picture of how the flow is behaving throughout the domain.

Applying transparency to the stream surfaces enables the user to view much more detail in terms what is happening within the flow at different levels and understand how the flow structure is developing. Also, it is very useful to use a colour scale showing the variation on various parameters e.g., velocity, pressure or turbulence intensity. This provides much more information to the user in the process of trying to identify and better understand the flow characteristics.

This framework is useful for the identification and visualisation of interesting flow features such as vortices, and for visualising the interaction of wakes behind obstructions with downstream objects falling within the wake's flow path. The tool would be of particular benefit to someone who may not be entirely familiar with the details or nature of the data.

CHAPTER 5

Stream Surface Seeding for a Land Speed Record Vehicle

THE challenge of visualising CFD simulations include handling large, unstructured, high dimensional data. Although many algorithms have been described for the seeding of streamlines, relatively few have been presented for stream surfaces. In this context, the work in Chapter 4 automating the seeding of stream surfaces is extended. We adapt a previous surface seeding algorithm such that it is more applicable to large, unstructured CFD simulation data with emphasis on reducing the memory requirements and computational time for the clustering, while processing unstructured data efficiently. Part of the motivation behind our work is a request from CFD experts to interactively select a subset (or cluster) of the CFD data and obtain more details. We further study the domain expert requirements, developing a visual interface and tools for interactive selection and filtering of cluster representations of CFD data.

The requirements of the domain experts include visualising large datasets, user guided semi automatic seeding of surfaces to represent interesting subsets of the flow, locating and visualising areas of turbulent flow. We define those large datasets as being able to just fit into RAM on standard CFD visualisation hardware (approx 8-16GB). Bloodhound SSC is a jet and rocket powered car designed for a speed of 1,000 mph (just over 1,600 kph). It is classified as a car because it has four wheels and is under full control of its driver. It has a slender body of approximately 14m length with two front wheels within the body and two rear wheels mounted externally within wheel fairings. It weighs over 7 tonnes and the engines produce more than 135,000 horsepower. The Bloodhound is a mix of car and aircraft technology, with the front half being a carbon fibre monocoque like a racing car and the back half being a metallic framework and panels like an

aircraft [Nob].

In order to accelerate computational speed, and process large datasets such as the Bloodhound SSC CFD simulation, we introduce an algorithm based on k-means clustering. k-means clustering is able to partition the flow into subsets of data defined by a customised distance function. To produce meaningful subsets comparable with the work in Chapter 4, we use a derived curvature field and a velocity gradient field both combined with Euclidean distance for a unified distance function. The distance function uses weighting parameters to enable the user to guide the final results.

Xu and Wunsch II [XW05] survey the topic of clustering, focusing on scalar clustering algorithms rooted in Statistics, Computer Science, and Machine Learning. Clustering algorithms can be divided into hierarchical clustering or partition-based clustering. The Telea and Van Wijk [TvW99] algorithm utilises a hierarchical approach, as does the work in Chapter 4. These algorithms are effective in providing a simplified representation of a vector field. However, these algorithms are $O(n^2)$ complex where n is the number of initial samples. This has a significant impact on computation and memory requirements. Alternatively, partitioning algorithms such as k-means clustering [Boc07] are generally $O(i \cdot k \cdot n)$ complex [Kog07], where k is the number of centroids or means, and i is the number of iterations. We utilise the k-means clustering to partition the domain using a novel distance function combining flow curvature, velocity gradient, and Euclidean distance. k-means is suitable for large, unstructured CFD datasets due to its computational efficiency and small memory footprint.

For sampling unstructured data we generally use a regular grid, Octree, BSP tree or other data structure storing pointers to intersecting tetrahedral cells, cutting down the search space when sampling the dataset. To improve memory usage and maintain good sampling speed we utilise spatial hash grids. Teschner et al. [THM*03] propose an approach to collision and self collision detection of dynamically deforming objects that consist of tetrahedrons. The algorithm employs a hash function for compressing a regular spatial grid. The hash function can be generated very efficiently and does not require complex data structures, such as Octrees or BSPs. The authors investigate and optimise the parameters of the collision detection algorithm, such as the hash function, hash table size and spatial cell size. Following this work Eitz et al. [EL07] present a hierarchical spatial hash grid scheme for real time collision detection. The authors employ an infinite hierarchical spatial hash grid in which for each single tetrahedron in the scene a well fitting grid cell size is computed. A hash function is used to project occupied grid cells into a finite 1D hash table. We employ this hierarchical spatial hash grid scheme for the fast, memory efficient sampling of an unstructured tetrahedral mesh.

To aid the capture of the underlying flow structures we use seeding curves derived from the curvature field. A seeding curve is placed at the centre of each cluster. This technique is used in Chapter 4, however we make a modification utilising an adaptive step integrator to refine the curve in complex areas resulting in a smoother stream surface. The clustering technique is used to locate areas of interest for the domain user, and then to generate seeding curves and surfaces associated with a given cluster, which yield

insightful representations of the CFD data.

Vector field clustering has been used to show interesting flow features for real world datasets [PGL*12]. It offers the benefit of not having to make a binary decision based on the presence of a feature. *k*-means clustering algorithms provide a general approach to partitioning data. This approach enables the domain expert to focus on an individual structure in the flow. We illustrate how to capture the characteristics of the flow field for the domain users. Our focus pays particular attention to the performance, memory footprint, and flexibility of the clustering. The main benefits and contributions of this chapter are:

- Improved computational speed and memory usage over recent stream surface work, with the ability to process large unstructured datasets fast and efficiently.
- An algorithm to partition the flow field using *k*-means clustering, providing superior performance with less memory overhead, and a feature-based overview of the data.
- A tailored algorithm specific to the bloodhound project which captures interesting subsets of the flow, while producing comparative results with previous hierarchical algorithms.
- The novel use of a curvature field and a velocity gradient field combined with Euclidean distance for the clustering distance function.
- The application of this visualisation and interaction techniques to real world CFD data with reviews from the domain experts.

The rest of this chapter is divided into the following sections. A detailed presentation of the algorithm is given in Section 2. The results are reviewed in Section 3. Domain expert feedback is provided in Section 4.

2 Stream Surface Placement

This section describes our adapted stream surface seeding algorithm, starting with an overview of the pipeline illustrated in Figure 5.1. The algorithm features clustering of combined Euclidean distance, curvature magnitude, and velocity gradient magnitude. Seeding curve generation starts from the cluster centres. We also discuss the weightings for the combined distance function and how they can be used to guide the results.

The input to our visualisation framework is an unstructured tetrahedral meshed CFD simulation. The input to the algorithm is $\mathbf{v}(\mathbf{p}) \in \mathbb{R}^3$, where $\mathbf{v} \in \mathbb{R}^3$, $\mathbf{p} \in \Omega$ and is an unstructured tetrahedral mesh in $\mathbb{R}^3$.

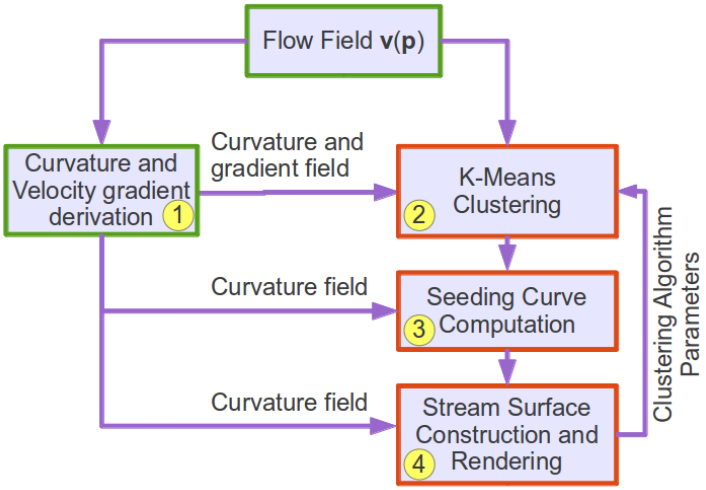


Figure 5.1: *The automated stream surface seeding pipeline. The pipeline shows the curvature field and velocity gradient field derived from the flow field. These are used as inputs to the clustering, seeding curve generation, and illustration techniques. Stream surfaces are propagated from the seeding curves through the vector field, and then rendered.*

1. We start by deriving a curvature field and a velocity gradient field. These fields are derived directly from the velocity field, and are used in the clustering process. The curvature field is also used to compute seeding curves after the clustering process has identified the seeding locations, and is used to map opacity to stream surfaces for illustrative rendering. These derived fields are saved to disk for quick loading by the user on subsequent analysis. See Section 2.1

2. We next partition the domain into k clusters where k is a user defined input. The data array indices of the vertices representing the scalar attributes are stored. The k-means algorithm iterates until it converges to a stable set of means. Flow curvature, velocity gradient, and Euclidean distance are comparison attributes that can be set. See Section 2.2.

3. Next we compute the seeding curves at the cluster centres. The cluster centre is automatically used as the basis of the seeding curve. The seeding curves are then generated by integrating forward and backward through the curvature field at a length proportional to the cluster size. This generates seeding curves which follow the local flow structures while maintaining orthogonality with the flow. See Section 2.3.

4. Stream surfaces are propagated from each of the seeding curves. Flow attributes may be mapped to colour and opacity. After the generation of the stream surfaces,

the surface data is rendered using a number of illustrative techniques to enhance the perception. See Section 2.4.

This pipeline differs from Chapter 4 in that we first compute the derived fields as input to the clustering process. We use the derived fields in combination with our customised distance metric to generate clusters in areas where there is a non zero velocity gradient and a concentration of clustering in areas of high curvature.

Following this we cluster the data with our k-means clustering algorithm rather than the hierarchical clustering proposed in Chapter 4. This reduces the computational and memory requirements, and enables the clustering of unstructured CFD data without modification. We then generate the seeding curves in the same way as in Chapter 4, modified to use an adaptive integration method. The motivation for this is that it increases the accuracy of the initial surface representation in complex areas.

Finally we construct and render the stream surfaces using the same illustration techniques as in the previous work in Chapter 4. These methods are effective for capturing the characteristics of the flow when applied to stream surfaces.

2.1 Derived Fields

From the vector field $\mathbf{v}(\mathbf{p})$ we derive a curvature field $\mathbf{c}(\mathbf{p})$, and a velocity gradient field $\mathbf{g}(\mathbf{p})$. The role of $\mathbf{c}(\mathbf{p})$ and $\mathbf{g}(\mathbf{p})$ is to support the clustering process (Section 2.2). $\mathbf{c}(\mathbf{p})$ also supports the seeding curve computation (Section 2.3) and illustrative techniques for rendering the stream surfaces (Section 2.4).

The curvature field $\mathbf{c}(\mathbf{p}) \in \mathbb{R}^3$, where $\mathbf{r} \in \mathbb{R}^3$, is derived by applying a combination of operators to the vector field. $\mathbf{c}(\mathbf{p})$ is derived from the velocity and acceleration of the flow field. Acceleration, $\mathbf{a}$, is defined as $\mathbf{a} = (\nabla \mathbf{v})\mathbf{v}$, where $\nabla \mathbf{v}$ is the Jacobian of the velocity field. Steady state curvature [Rot00] is defined as $|\mathbf{c}|$ where:

$$\mathbf{c} = \frac{\mathbf{v} \times \mathbf{a}}{|\mathbf{v}|^3} \tag{5.1}$$

The velocity gradient field $\mathbf{g}(\mathbf{p})$, where $\mathbf{g} \in \mathbb{R}^3$, is derived from the velocity field as follows:

$$\nabla |\mathbf{v}| = \frac{\partial |\mathbf{v}|}{\partial x} i + \frac{\partial |\mathbf{v}|}{\partial y} j + \frac{\partial |\mathbf{v}|}{\partial z} k \tag{5.2}$$

where $\mathbf{g} = \nabla |\mathbf{v}|$ i.e. the gradient of the velocity magnitude, and i, j, k are the components of a unit vector. The gradient field is computed to support the clustering process (Section 2.2).

2.2 k-means Clustering

k-means clustering is a fast, simple, and popular method for clustering data. With relatively low computational requirements and memory usage compared to hierarchical

clustering, it is a good candidate to solve data partitioning and significantly reduces the computational and memory requirements. This algorithm needs no modification to deal with either structured or unstructured data.

k-means is one of the simplest unsupervised learning algorithms that solves the well known clustering problem [Boc07]. The procedure provides a way to classify a given data field into k clusters chosen as a priori by the user. The main idea is to define k centroids, one for each cluster. We refer the reader to Section 3 for a discussion of the choice of k. The next step is to take each grid vertex $\mathbf{p}_i$ belonging to Ω and associate it with the nearest initial centroid.

For a set of scalars $\mathscr{S} = \{s(\mathbf{p}_1), ..., s(\mathbf{p}_n)\} \subset \mathbb{R}$, a distance function $d(c,\ s(\mathbf{p}_i))$ defines a centroid $c = c(\mathscr{S}) \subset \mathbb{R}$ of the set $\mathscr{S}$ as a solution of the minimisation problem [Kog07]:

$$c = arg\ min\left\{ \sum_{s(\mathbf{p}_i)\in\mathscr{S}} d(c,\ s(\mathbf{p}_i)) \right\} \tag{5.3}$$

Let $\prod = \{\pi_1, ..., \pi_k\}$ be a partition of $\mathscr{S}$, that is:

$$\bigcup_i \pi_i = \mathscr{S},\ \text{ and }\ \pi_i \cap \pi_j = 0\ \text{ if }\ i \neq j \tag{5.4}$$

Given a partition $\prod = \{\pi_1, ..., \pi_k\}$ of the set $\mathscr{S}$ one can define the corresponding centroids $\{c(\pi_1), ..., c(\pi_k)\}$ by:

$$c(\pi_i) = arg\ min\left\{ \sum_{s(\mathbf{p}_i)\in\pi_i} d(c,\ s(\mathbf{p}_i)) \right\} \tag{5.5}$$

For a set of k centroids $\{c_1, ..., c_k\}$ one can define a partition $\prod = \{\pi_1, ..., \pi_k\}$ of the set $\mathscr{S}$ by:

$$\pi_i = \{s : s \in \mathscr{S},\ d(c_i,\ s) \leq d(c_j,\ s) \text{ for each } j = 1, ..., k\} \tag{5.6}$$

noting that in general $c(\pi_i) \neq c_i$. The procedure iterates between the two steps described above until it converges to a stable set of means and partitions. It is important to note that we perform our clustering in scalar space utilising the customised distance metric described next.

A Customised Distance Metric Chapter 4 guides the clustering process to areas of flow where there is a change in geometric curvature e.g. difference in velocity vector direction (η_δ), and velocity gradient (η_μ). These attributes combined with position (η_ψ) form the basis of the distance function. The distance function can be altered by modifying the bias between each of the three components.

The aim is to generate seeding curves in locations adjacent to the flow structures, with minimal seeding density. Interesting flow structures may be characterised by their

curvature and velocity gradients. Another aim is to generate clusters in areas of non zero velocity gradient and a concentration of clustering in areas of high curvature.

Based on the same philosophy we introduce a novel distance function $d(c(\pi),\ s(\mathbf{p}))$ combining flow curvature $|\mathbf{c}|$, velocity gradient $|\mathbf{g}|$ and Euclidean distance $|\mathbf{e}|$, where $\mathbf{e} \in \mathbb{R}^3$, for use with k-means clustering.

To incorporate the curvature of the flow, velocity gradient, and Euclidean distance into a single unified distance function we specify their relationship as a simple linear combination:

$$l'(\ |\mathbf{c}|, |\mathbf{g}|, |\mathbf{e}|\ ) = |\mathbf{c}| + |\mathbf{g}| + |\mathbf{e}| \tag{5.7}$$

This combination alone does not enable flexibility for fine tuning the results in marginal cases where either flow curvature or velocity gradient strongly influences the results. Additionally we desire the ability to produce both focus and contextual visualisations for the domain user. This motivates us to add user specified bias to the distance function. To control the influence of $|\mathbf{c}|$ or $|\mathbf{g}|$ we specify a linear relationship:

$$i(\ |\hat{\mathbf{c}}|, |\hat{\mathbf{g}}|\ ) = B|\hat{\mathbf{c}}| + (1-B)|\hat{\mathbf{g}}| \tag{5.8}$$

where $B \in [0,1]$ and $\hat{\mathbf{c}}$ and $\hat{\mathbf{g}}$ are range normalised e.g. $|\hat{\mathbf{c}}| \in [0,1]$ and $|\hat{\mathbf{g}}| \in [0,1]$. To influence a focus or contextual bias we specify a linear relationship between i and $|\hat{\mathbf{e}}|$ where $A \in [0,1]$ and $\hat{\mathbf{e}}$ is normalised by the domain extents e.g. $|\hat{\mathbf{e}}| \in [0,1]$:

$$l(\ |\hat{\mathbf{c}}|, |\hat{\mathbf{g}}|, |\hat{\mathbf{e}}|\ ) = Ai(\ |\hat{\mathbf{c}}|, |\hat{\mathbf{g}}|\ ) + (1-A)|\hat{\mathbf{e}}| \tag{5.9}$$

The distance function $d(c(\pi),\ s(\mathbf{p})) = ||c(\pi) - s(\mathbf{p})||^2$ determines how we partition the domain. Both the centroid $c(\pi)$ and the attribute $s(\mathbf{p})$ are scalar values computed from the linear relationship $l(\ |\hat{\mathbf{c}}|(\mathbf{p})\ ,\ |\hat{\mathbf{g}}|(\mathbf{p})\ ,\ |\hat{\mathbf{e}}|(\mathbf{p})\ )$. By changing the parameters A and B we can communicate different insights into the characteristics of the flow behaviour. For example, increasing the value of A will provide a more focused visualisation, conversely,

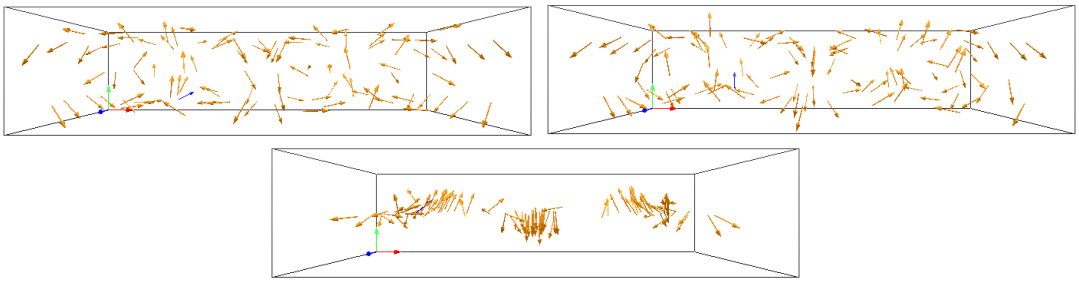


Figure 5.2: *This image shows the Bernard flow simulation clustered with our algorithm. The cluster centres are represented by the bases of the arrow glyphs where each glyph shows vector direction at that location. All images are clustered with B = 0.5, and k = 100. The top left image is clustered with parameter A = 0.05, and shows a more evenly distributed set of clusters as we emphasise Euclidean distance. The top right image is clustered with A = 0.5, and the bottom image is clustered with A = 0.95 showing the emphasis moving towards the centre of the curved flow with higher values of A.*

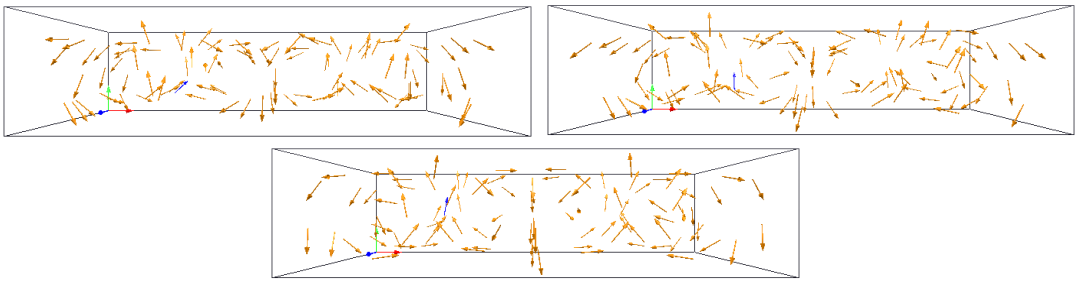


Figure 5.3: *The cluster centres are represented by the bases of the arrow glyphs where each glyph shows vector direction at that location. All images are clustered A = 0.5, and k = 100. The top left image is clustered with parameter B = 0.1. The top right image is clustered with B = 0.5, and the bottom image is clustered with B = 0.9. The changes are more subtle when emphasising velocity gradient over flow curvature.*

reducing A will provide increased context to the visualisation. Adjusting B will adjust the influence either $|\mathbf{c}|$ or $|\mathbf{g}|$ has toward the final visualisation.

User Input parameters The choice of user input parameters A, B, and k will effect the resulting visualisation. In this section we show the effect the different parameters have on the distribution of the clusters. It can be seen in Figure 5.2 that as the parameter A is decreased it provides a more evenly distributed set of clusters which provide a good overview of the characteristics of the flow field. In contrast it can be seen that an increase of A results in a more focused distribution of clusters in areas of higher curvature.

The change in emphasis between flow curvature and velocity gradient enables the user to fine tune the clustering, adjusting the cluster centres for more favourable results. We demonstrate the effects of changing parameter B in Figure 5.3.

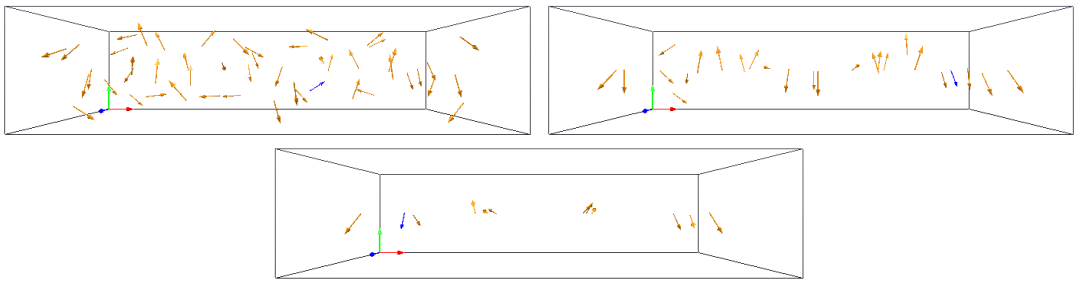


Figure 5.4: *These image shows the Bernard flow simulation clustered with our algorithm. The cluster centres are represented by the bases of the arrow glyphs where each glyph shows vector direction at that location. All images are clustered A = 0.5, and B = 0.5. The top left image is clustered with a k of 50, the top right with a k of 25, and the bottom image with a k of 12. As the quantity of clusters reduce a more focused distribution is observed.*

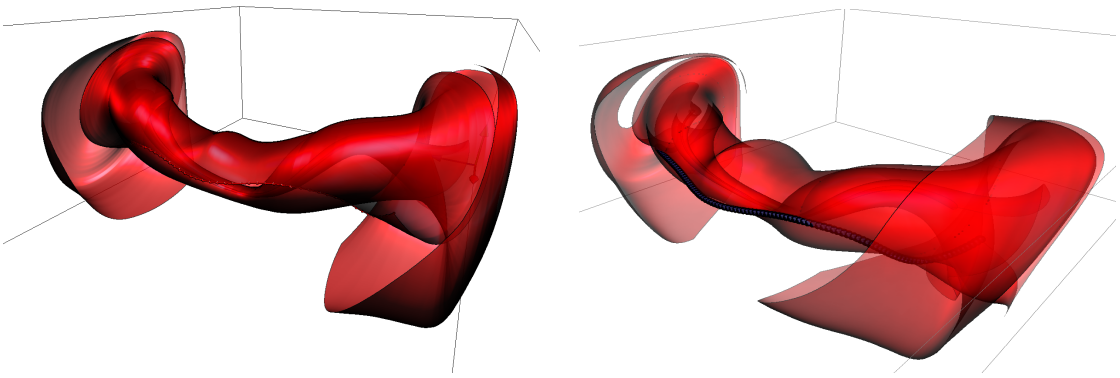


Figure 5.5: *The left illustration of the Bernard simulation demonstrates the seeding curve following the flow structure. The seeding location is derived from the clustering with parameters $A = 0.5$, $B = 0.5$, and $k = 4$. Three of the surfaces are hidden in the rendering. The right illustration of the Bernard simulation is provided for comparison courtesy of Edmunds et al. [ELM*12].*

Another important parameter is the choice of k. Too many clusters will produce a cluttered visualisation, and too few will not adequately capture the interesting features within the flow field. The effect of changing k can be seen in Figure 5.4. The problem of removing the need to choose k, is present for the general case of k-means clustering. There is much literature on this matter as discussed in Fang et al. [FW12] and Ferreira et al. [FKSS12], however we are not aware of any definitive approach to solving this problem. We therefore demonstrate our visualisations, highlighting the number of chosen k, to provide guidance to the domain engineer. k is a user option.

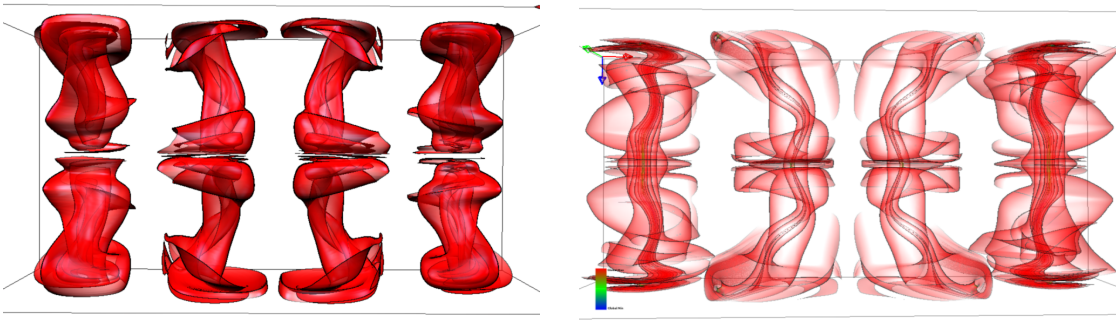


Figure 5.6: *The left illustration is the Bernard flow numerical simulation visualised using our seeding algorithm. The seeding locations are derived from the clustering process using parameters $A = 0.4$, $B = 0.5$, and $k = 8$. The four main double vortex structures are clearly emphasised by the eight surfaces. The thermal motion of the flow field is captured with our framework. The figure shows the surfaces rendered with transparency. The right illustration of the Bernard simulation is provided for comparison from Chapter 4*

2.3 Seeding Curve Computation

The seeding curve generation commences following the clustering. The final set of k centroids $\{c_1, ..., c_k\}$ are used to define the location for the origin of each seeding curve. Seeding curves are generated using the same method as in [ELM*12]. The seeding curves are generated by integrating through the curvature field. Their length are restricted to the cubic root of the cluster volume:

$$length(\pi) = \sqrt[3]{vol(\pi)} \tag{5.10}$$

where $vol(\pi)$ is the spatial volume of the cluster $\pi(c_k)$. The integration is achieved using a fourth-order Runge-Kutta integrator, sampling the curvature field, in forward and backward directions. We modify the original approach by using an adaptive fourth-order Runge-Kutta integrator, which provides a discretised seeding curve with a denser set of vertices in areas of increased complexity. The motivation for this is that it increases the accuracy of the surface representation in complex areas.

2.4 Surface Construction and Rendering

For the sampling of unstructured tetrahedral data we use a hierarchical spatial hash grid as described in Appendix A. We employ this method for fast lookup of vertex to tetrahedral cell intersection and interpolation. We utilise the hierarchical spatial hash grid which for each single tetrahedron in the domain, a well fitting grid cell size is computed.

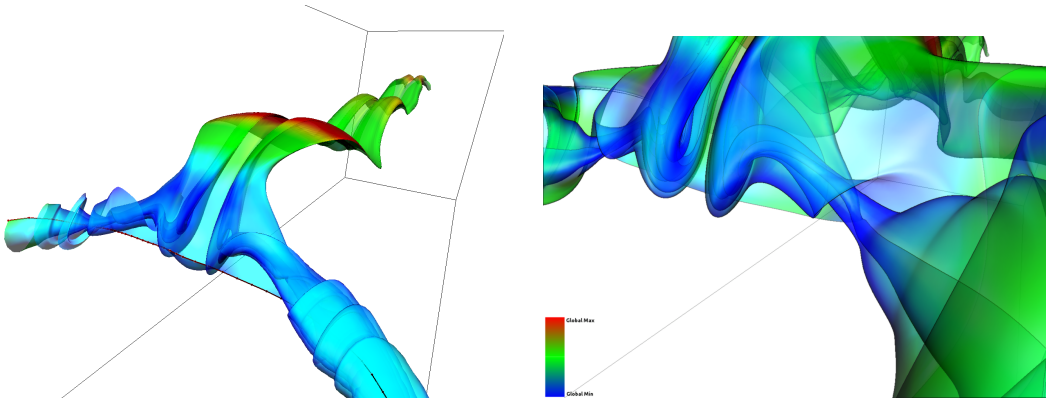


Figure 5.7: *The dataset shown here is a direct numerical Navier Stokes simulation by Simone Camarri and Maria Vittoria Salvetti (University of Pisa), Marcelo Buffoni (Politecnico of Torino), and Angelo Iollo (University of Bordeaux I) [CSBI05] which is publicly available [Tos]. We use a uniformly resampled version which has been provided by Tino Weinkauf and used in von Funck et al. for smoke surface visualisations [vFWTS08a]. The left illustration of flow past a cuboid simulation demonstrating a stream surface generated near a double vortex structure emanating from a critical point. The clustering parameters used for this visualisation are $A = 0.9$, $B = 1.0$, and $k = 3$. The right illustration of the cuboid simulation is provided for comparison from Chapter 4.*

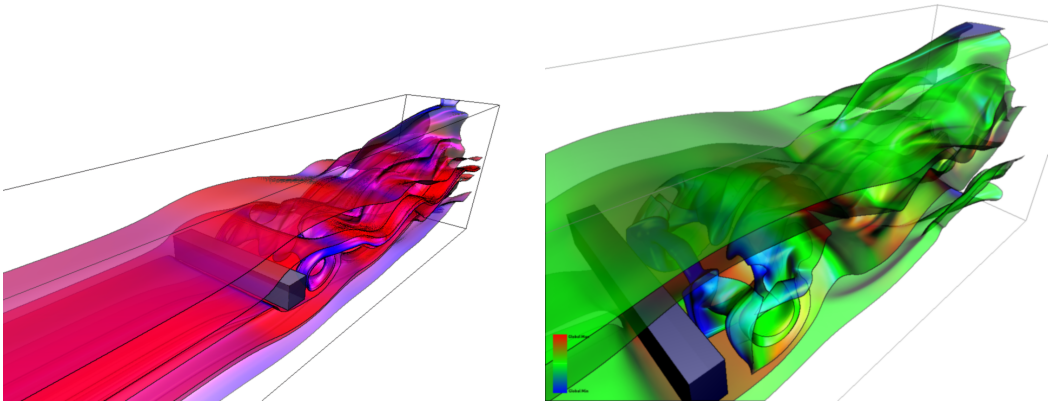


Figure 5.8: *The right image of the flow past a cuboid simulation highlights the ability of our algorithm to provide both focus and context within the same visualisation. The vortex shedding is represented by the stream surfaces along with its relation to the rest of the flow field. The image is rendered with transparency. The clustering parameters used for this visualisation are $A = 0.5$, $B = 0.5$, and $k = 9$. The left illustration of the cuboid simulation is provided for comparison Chapter 4.*

A hash function is used to store a reference to the occupied grid cells in a finite 1D hash table. The hash is computed from the bounding box of the tetrahedral cell. When performing lookup, the hash is computed from the sample vertex. This method reduces the memory footprint for cell storage, while maintaining fast lookup for intersection and interpolation tests. In the case of the full Bloodhound SSC CFD data the resolution of an equivalent regular grid, fine enough to capture a similar quantity of tetrahedral cells per grid location, would be $13,607 \times 13,607 \times 6,792$. At one 64bit pointer per grid cell, an approximate memory usage would be 9,000 GB, not including any other storage overhead. However, the hierarchical spatial hash grid, at one pointer and one integer per tetrahedral cell, uses approximately 500 MB not including any other storage overhead. The storage overhead for the hierarchical spatial hash grid relies on a hash map / binary tree implementation for fast lookup. This is more expensive than a simple array for the regular grid. Using standard C++ STL map containers the memory overhead rises to approximately 2.5 GB.

Our work utilises an out of the box solution for generating stream surfaces [GKT*08]. An adaptive fourth-order Runge-Kutta integrator (See Appendix A) is used in the surface construction. The user can select downstream and upstream propagation. Surfaces are terminated when they leave the domain, enter a periodic orbit, or reach a predetermined maximum length. The user has an option to control the length.

A number of techniques are implemented to aid the viewer in perception of the resulting visualisation. Options include the use of transparency, colour, and silhouette edge highlighting. Transparency in visualisations pose problems relating to the order of primitive rendering. We use depth peeling for order independent transparency. Silhouette edge highlighting is used to help the viewer in perceiving where the surfaces curve away from the viewer, and enhance surface edges. Silhouette highlighting utilises

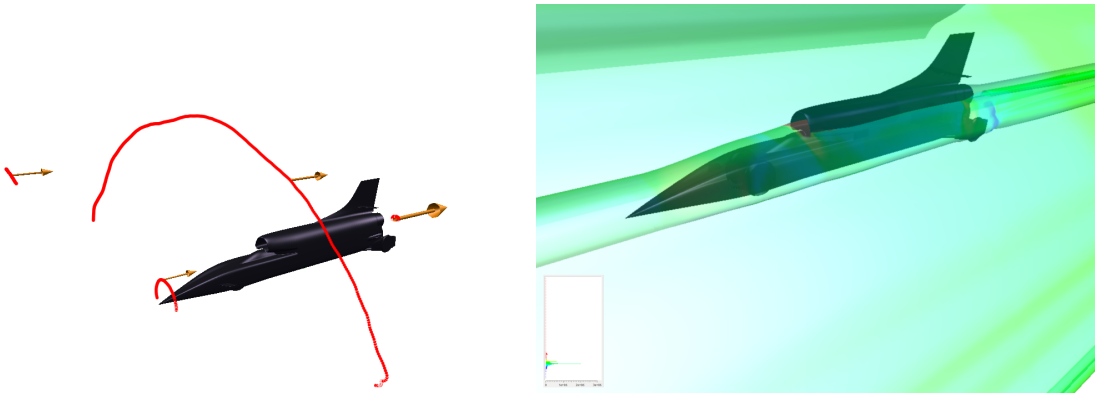


Figure 5.9: *Bloodhound SSC CFD simulation clustered using parameters $A = 0.1$, $B = 0.5$, and $k = 5$. The left visualisation demonstrates seeding curves generated from the cluster centres by integration through the curvature field. The cluster centres are represented by arrow glyphs where the arrow base is the centre, and the glyph represents the velocity vector at that location. The right visualisation demonstrates stream surfaces generated from the seeding curves in Figure 5.9. The surfaces are rendered using illustrative techniques, and colour mapped to the coefficient of pressure c_p clamped to the range [-1,6]. Red is high pressure and blue is low pressure as compared to the free stream pressure which is green.*

a Gaussian kernel in image space [MH02]. Reducing the saturation of colour as the surfaces curve away from the viewer further enhances the perception of shape, as Chapter 4.

3 Results

Interaction and exploratory tools are very useful to the domain expert for the production of insightful and meaningful visualisations. Our clustering approach partitions the domain into meaningful subsets. Placing stream surfaces in areas of interest provides feedback about the underlying characteristics of the flow. In the following section we compare our technique to Chapter 4, utilising the same datasets. We then demonstrate our algorithm in a case study of the Bloodhound CFD simulation from domain experts for which our algorithm is tailored.

3.1 Visual Comparisons

In this section we compare our algorithm to previous work in Chapter 4. We demonstrate how we are able to capture the same features found within each of the datasets. We first look at the Bernard flow numerical simulation defined on a regular grid [WSE05]. The simulation demonstrates thermal motion as a result of convection. Our algorithm captures the double vortex structures using parameters $A = 0.5$, $B = 0.5$, and $k = 4$. As shown in Figure 5.5, we have illustrated one of the double vortex structures, which com-

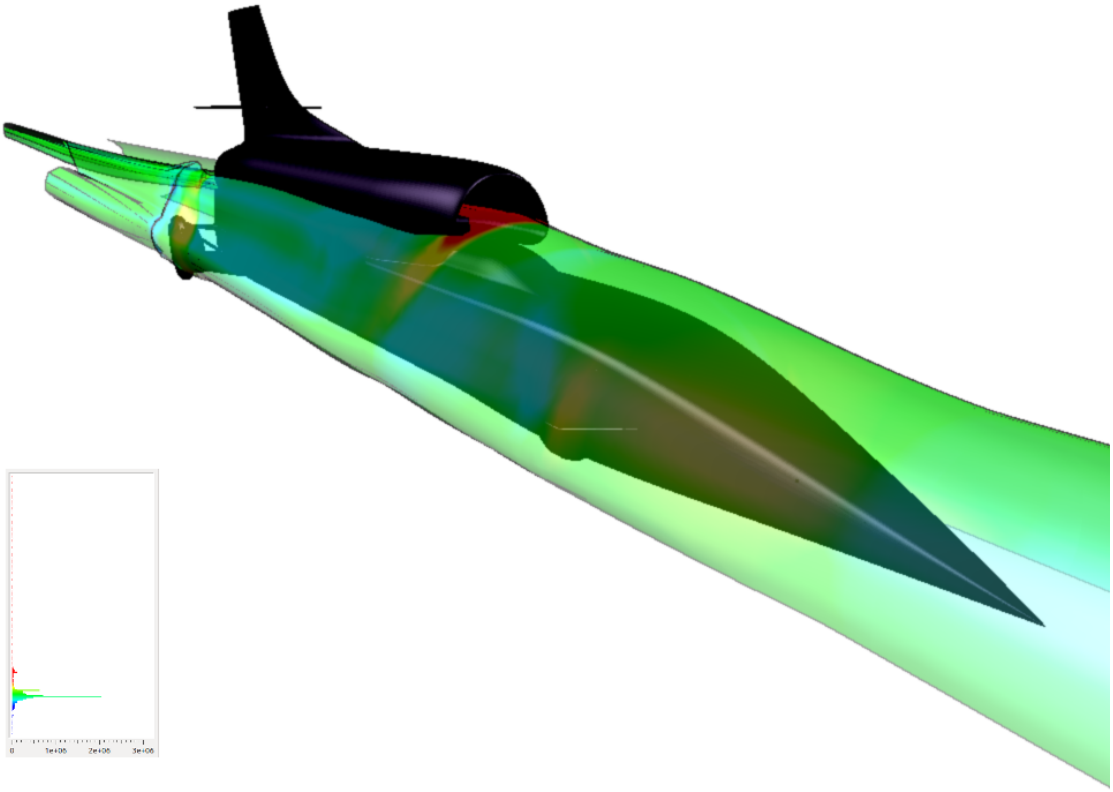


Figure 5.10: *Bloodhound SSC CFD simulation clustered using parameters $A = 0.1$, $B = 0.5$, and $k = 5$. This visualisation demonstrates stream surfaces generated from the seeding curves in Figure 5.9. The contextual surfaces are hidden in this visualisation. The surfaces are rendered using illustrative techniques, and colour mapped to the coefficient of pressure c_p clamped to the range [-1,6]. Red is high pressure and blue is low pressure as compared to the free stream pressure which is green. A colour map histogram is shown in the bottom left of the image.*

pares well with Figure 4.10 from Chapter 4. We also show a comparative visualisation of the full Bernard illustration as shown in Figure 5.6. This visualisation is generated from parameters $A = 0.4$, $B = 0.5$, and $k = 8$. This visualisation compares well with Figure 4.16 from Chapter 4, and has the advantage of seeding each side of the double vortices separately. This is useful for a more detailed exploration of the flow structures.

Next we look at the flow past a cuboid simulation as can be seen in Figure 5.7. This visualisation demonstrates the ability of our algorithm to capture the double vortex structure down stream of the cuboid. In Figure 5.8 we demonstrate the visualisation of both focus and context using one set of cluster parameters $A = 0.5$, $B = 0.5$, and $k = 9$. Figure 4.15 from Chapter 4 was generated using two sets of parameters (focus and context settings). In summary our algorithm compares well with the approach of Chapter 4. We are able to capture the same underlying characteristics of the flow fields with improved computational speed while maintaining a smaller memory footprint.

In addition to the visual comparison with the work in Chapter 4 we present a gallery of visualisations demonstrating the effect of changing the parameters A, B, and k, for a given dataset. This gallery is presented in Appendix B.

3.2 Memory and Performance Evaluation

To evaluate our work, we compare the proposed method with the work in Chapter 4. Our algorithm performs well when compared to the hierarchical clustering of Chapter 4. The hierarchical clustering is $O(n^2)$, compared with our algorithm whose complexity is $O(i \cdot k \cdot n)$, where n is the number of initial samples. It is important to note that no parallelisation techniques are used in our evaluation. This is to provide clarity of the algorithm performance. Parallelisation techniques can vary significantly, and optimisation of these techniques for speed or memory usage can disguise the baseline performance of our technique.

Our algorithm is implemented in C++ and QT4 on a PC with an NVIDIA GeForce GTX480, an Intel quad core 2.8GHz CPU and 8GB RAM. The data is loaded using either the freely available TecplotIO library [Tec], or by loading the Swansea University FLITE [Swab] CFD simulations directly. The bottleneck in the performance of our algorithm is the clustering as seen in Table 5.1. The performance of stream surface rendering is comparable to previous work e.g. Born et al. [BWF*10], Hummel et al. [HGH*10], and Chapter 4. The memory requirement for our approach is proportional to the size of

Clustering Performance				
Data	Elements	k	Time[ms]	Chapter 4[ms]
Bernard	$128x32x64$	4	1,246	2,908
Cuboid	$192x64x48$	3	2,095	6,898
Lorenz	128^3	5	11,058	34,812
Air brake	5,764,071	10	10,795	n/a
Bloodhound	40,963,951	5	37,011	n/a

Table 5.1: *Clustering performance of a range of simulations. All are regular grid data except the Bloodhound data, which is an unstructured tetrahedral grid. For reference the last column contains comparative times from Chapter 4.*

the data set e.g. n, where n is the number of initial samples. We store only one integer per vertex to reference the cluster it belongs to. The hierarchical approach stores vector data, location data, cluster size data, cumulative error data, and neighbourhood/connectivity information at each node in the binary tree. The quantity of nodes is $2n-1$. The k-means clustering with the customised distance metric approach can reach up to one order of magnitude faster than its predecessor.

3.3 Case Study: Bloodhound Project

In our case study we illustrate the results of our algorithm applied to the Bloodhound SSC CFD data. This demonstrates the visualisation of large CFD datasets, user guided semi automatic seeding of surfaces to represent interesting subsets of the flow, and the ability to locate and visualise areas of turbulent flow which may slow down or interfere with the stability of the vehicle. In general terms the engineers are looking for flow phenomena which may interfere with the vehicle's aerodynamics, causing the vehicle to become uncontrollable and crash. The engineers are also searching for flow phenomena which may cause drag thus hindering the record attempt.

Our study is divided into two subsections. The first is a study of the Full CFD simulation where we are trying to locate any areas of turbulent flow along the length of the vehicle. The second is a study of the turbulent flow which forms behind the air brakes when deployed.

Full Simulation at Mach 1.3

The CFD data representing the Bloodhound SSC travelling at Mach 1.3 is a large unstructured tetrahedral mesh consisting of 40,963,951 elements. The simulation results include velocity $(\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z)$, density (ρ), coefficient of pressure (c_p), and energy (E).

We choose a set of parameters that provides a reasonably distributed set of surfaces, while maintaining proximity to the vehicle body, with the aim of capturing the flow characteristics around the vehicle in transit. The clustering results from parameters $A = 0.1$, $B = 0.5$, and $k = 5$ are illustrated in Figure 5.9. We are searching for any indication of turbulent flow forming along the boundary layer of the vehicle body.

Figure 5.9 demonstrates the surfaces generated from the seeding curves and rendered with illustrative techniques. In Figure 5.10 the contextual surface is hidden to allow closer examination of the flow along the boundary of the vehicle. These illustrations help confirm the aerodynamic characteristics, showing no indication of unexpected areas of turbulent flow. The coefficient of pressure is colour mapped to identify any undesirable pressure spikes forming as a result of shock waves traversing the body.

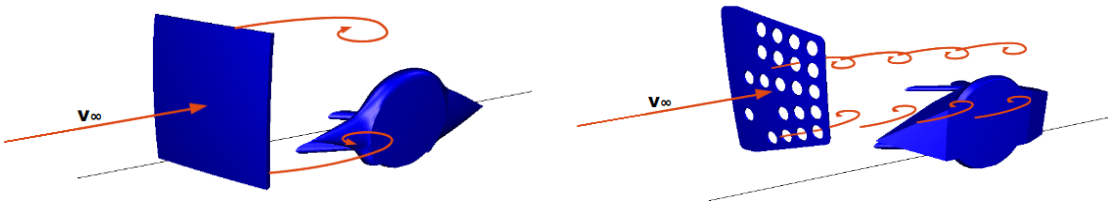


Figure 5.11: *Bloodhound SSC CFD simulation geometry. The left image shows the initial air brake design configuration with a solid construction, situated just in front of the rear suspension. The right image shows the final air brake design configuration with holes, again situated just in front of the rear suspension.*

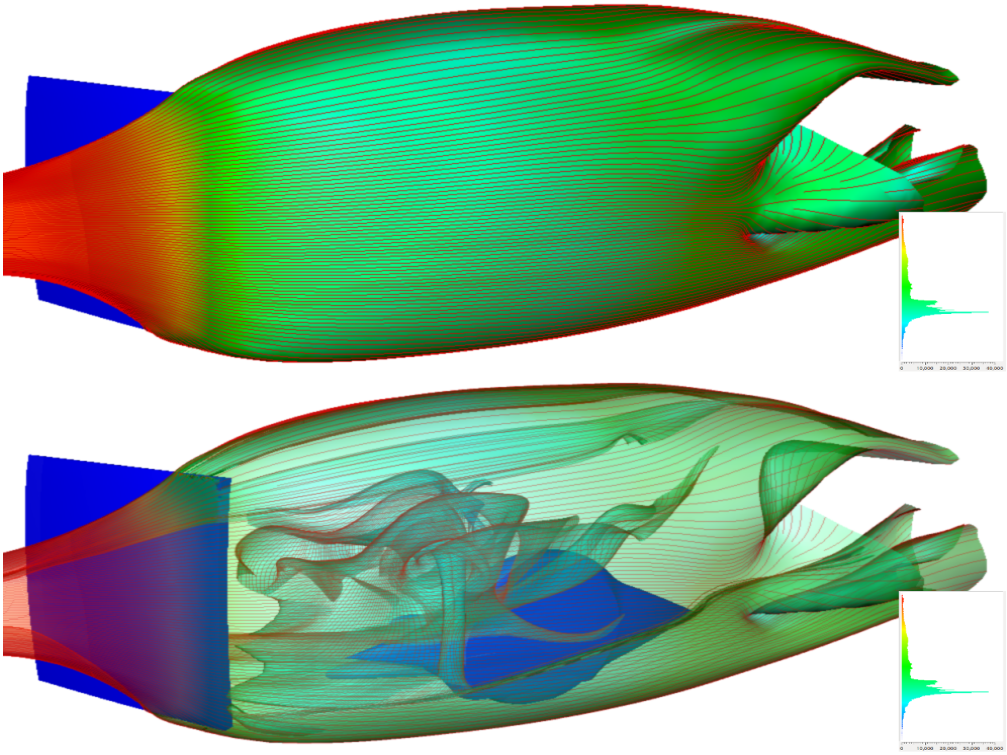


Figure 5.12: *Bloodhound SSC CFD simulation of the the initial air brake design configuration in use, clustered using parameters $A = 0.6$, $B = 0.6$, and $k = 13$. The bottom image is using transparency to aid the visual perception of the flow behind the air brake. A large vortex structure can be clearly seen left of centre. Colour is mapped to coefficient of pressure c_p clamped to the range [-1,6], where free stream c_p is green, high c_p is red, and low c_p is blue.*

Simulation of Air Brakes

The CFD data representing the Bloodhound SSC deployed air brakes is a large unstructured tetrahedral mesh consisting of 5,764,071 elements. The simulation results include velocity ($\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z$), density ($\rho$), coefficient of pressure (c_p), and energy (E). In this case study we investigate two simulations; the first is the initial design configuration of the air brake which is a solid 'door' situated up stream of the rear suspension assembly, the second is a revised air brake design which includes a set of holes. It is predicted the first air brake design will shed large, high intensity vortex structures at a frequency close to the natural frequency of the rear suspension assembly. The revised design includes a set of holes which are predicted to produce a much higher frequency of smaller vortex structures which will greatly reduce the interference with the rear suspension assembly. See Figure 5.11 for an image of the geometry used for these CFD simulation. This figure also demonstrates a simplified overview of the expected results.

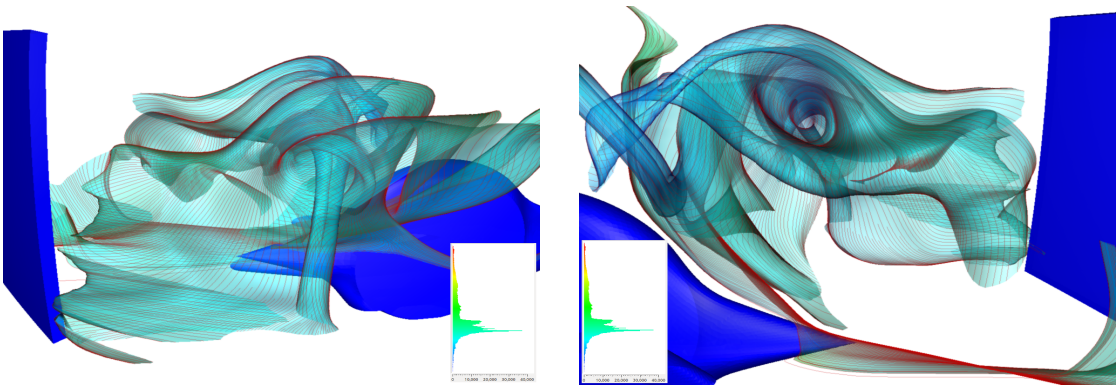


Figure 5.13: *The left visualisation shows a close up of 5.12 with the outer surface hidden. The large vortex structure can clearly be seen at the centre of this image. The right visualisation views the same vortex structure from behind. The throat of the vortex is located above the centre. Two further vortex structures can be seen; one rolling around the top of the throat, and one rolling around the bottom. Colour is mapped to coefficient of pressure c_p clamped to the range [-1,6], where free stream c_p is green, high c_p is red, and low c_p is blue.*

For the first design simulation we choose a set of parameters which focuses on areas of high curvature with the aim of capturing large, high intensity vortex structures behind the air brake mechanism. The clustering results from parameters $A = 0.6$, $B = 0.6$, and $k = 13$ are illustrated in Figures 5.12 and 5.13. In these visualisations we can confirm the presents of a large, high intensity vortex structure travelling towards the rear suspension assembly. There is only one instance of a vortex structure in the space between the air brake and past the rear suspension, indication a lower frequency of vortex shedding. This visualisation of the CFD simulation clearly confirms the engineers initial predictions.

For the second design simulation we adjusted the parameters focus on a higher density of interwoven curved flow structures forming behind the air brake mechanism. We increased the number of k to achieve this, and increased the parameter A to further concentrate the clustering in curved areas as we expect a reduction of extreme curvature found on the first simulation. The clustering results from parameters $A = 0.95$, $B = 0.6$, and $k = 19$ are illustrated in Figures 5.14, 5.15 and 5.16. It can be seen that the flow behind the air brake is highly complex. As expected this is a result of the hole configuration causing the formation of small vortices being shed at high frequency from each of the holes. The high frequency of these low intensity vortices quickly become interwoven into a highly complex area of turbulent flow. This flow pattern again confirms the engineers predictions and provides great insight into how the turbulent flow is forming and interacting with the suspension assembly and bodywork.

These illustrations support the domain engineer in consolidating their understanding of the predicted flow characteristics, and in understanding how the air brake design may need to be modified to further improve vehicle stability under braking.

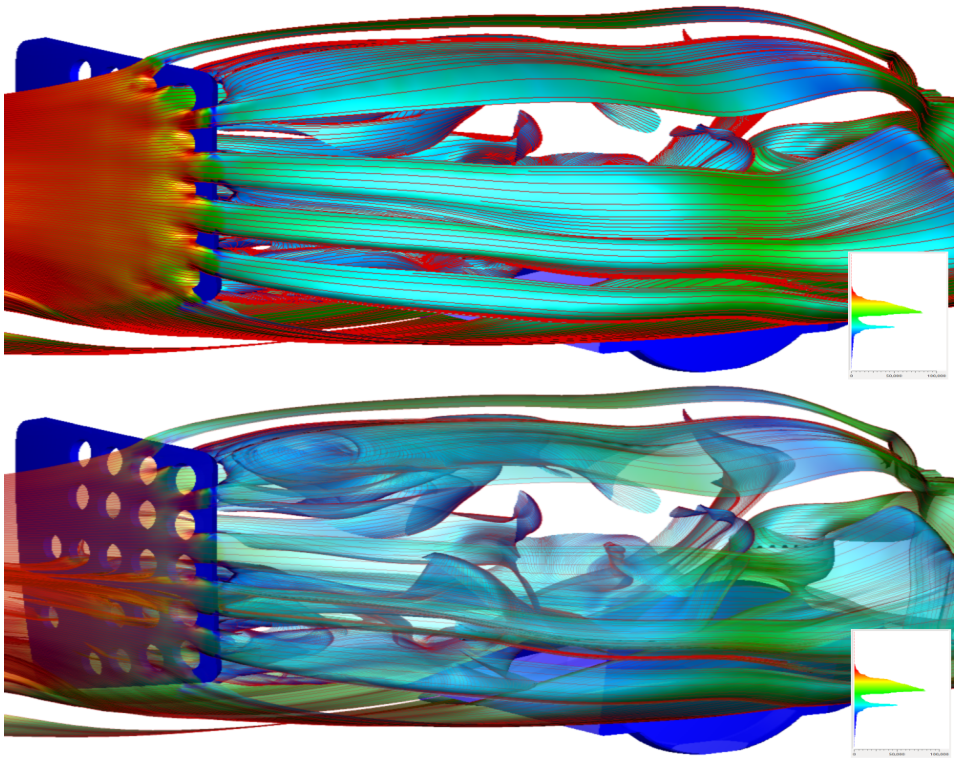


Figure 5.14: *Bloodhound SSC CFD simulation of the the final air brake design configuration in use, clustered using parameters $A = 0.95$, $B = 0.6$, and $k = 19$. The bottom image is using transparency to aid the visual perception of the flow behind the air brake. It can be seen that the flow behind this version of the air brake is highly complex. Colour is mapped to coefficient of pressure c_p clamped to the range [-1,6], where free stream c_p is green, high c_p is red, and low c_p is blue.*

4 Domain Expert Feedback

The work presented in this chapter is undertaken in close collaboration with domain engineers for the purpose of developing an effective application for the improved analysis of flow phenomena. As a result of the work undertaken, Dr Ben Evans provides us with valuable expert feedback which follows. Dr Ben Evans is a member of the Bloodhound SSC design team, and is employed as a researcher in the department of Aerospace Engineering at Swansea University.

Effective and useful flow visualisation for complex geometrical applications such as the Bloodhound SSC vehicle is notoriously challenging. Industrial CFD applications, such as Bloodhound, have specific problems associated with them in terms of flow visualisation including high computational mesh resolution and high levels of adaptivity in the mesh. Full vehicle Bloodhound CFD meshes, for example, typically contain approximately 10 million sampling points (and greater) with cell sizes varying from $10e^{-5}$ m

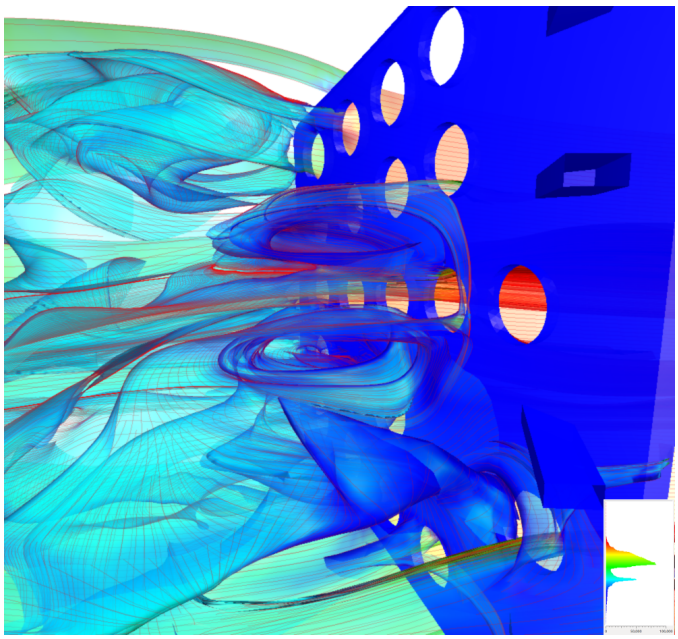


Figure 5.15: *The visualisation shows a close up of 5.14 viewed from the other side. Two small vortex structures can clearly be seen at the centre of this image, forming just behind the air brake. Colour is mapped to coefficient of pressure c_p clamped to the range [-1,6], where free stream c_p is green, high c_p is red, and low c_p is blue.*

within the highly refined boundary layer regions in contact with viscous surfaces to over 1m in the far field domain. Bloodhound SSC, being a Land Speed Record vehicle, has the additional complexity of supersonic flow in close proximity to a large ground surface. This leads to the requirement by the CFD engineer of an understanding of the interaction of shock waves with the ground plane and vehicle. Industrial CFD simulations are also typically used within design cycles that require CFD engineers to make rapid design cycle decisions. An efficient and robust system for interpreting the vector field resulting from the CFD analysis is therefore highly desirable.

The behaviour of aerodynamic bodies such as Bloodhound SSC is governed by the force coefficients that they generate at a given flight condition, e.g. lift coefficient as a function of Mach number and angle of attack, or drag coefficient as a function of Mach number and air brake deployment position. However, it is often very difficult to determine why a force coefficient varies from one design configuration to another using the traditional flow visualisation methods of pressure coefficient plots over body surfaces or cuts through the domain, or glyphs at the computational sampling points (for the reasons already mentioned). An ability to relate the variation of force coefficients (and the position of their application on the body such as centre of pressure) to the underlying causal flow phenomena such as flow separation, vortex shedding and shock wave formation is important in making informed design decisions in the engineering design cycle.

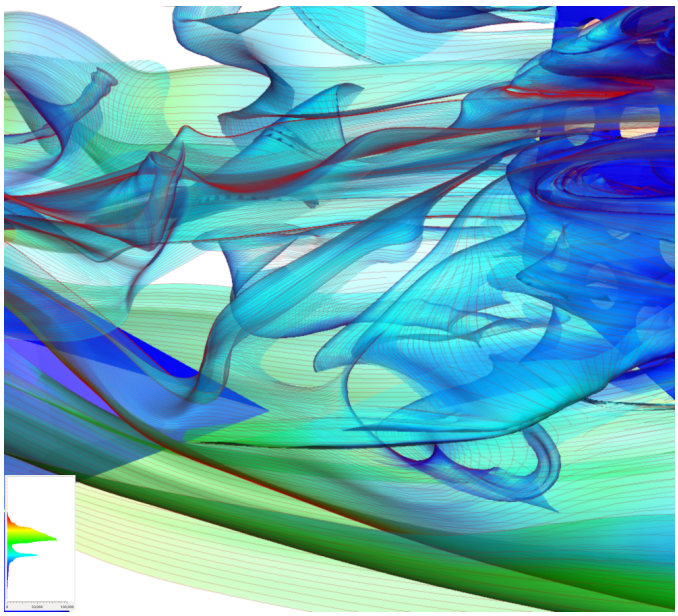


Figure 5.16: *The visualisation is a view to the left of 5.15. The two small vortex structures, identified in Figure 5.15, can be seen in the upper right of this image. Following a line from the bottom right towards the top left, four small vortex structures can be seen moving away from the air brake up and over the top of the rear suspension assembly. Colour is mapped to coefficient of pressure c_p clamped to the range [-1,6], where free stream c_p is green, high c_p is red, and low c_p is blue.*

The approach presented in this chapter goes a significant way towards eliminating both of these issues: speed and consistency in analysis. Note that there is a great benefit that the user still has the ability to interact with the visualisation algorithm via the parameters A, B, and k and maximum surface length in order to tune the resulting visualisation, but holding these parameters constant whilst comparing two designs provides a level of consistency often not available when the user is manually determining the seeding positions. The automatic seeding of the stream surfaces from cluster centres ensures that for a given vector field dataset there is the maximum probability of capturing the most important/significant features of the flow.

The two Bloodhound case studies used to test the approach detailed in this chapter provide excellent tests for the usefulness of the approach. In the first case, of supersonic flow over the full Bloodhound vehicle with the air brakes retracted, it is anticipated that flow separation and vortex shedding should be at an absolute minimum since this results in the minimum drag solution. In contrast, the second case considers the subsonic behaviour of the fully extended air brake.

Two air brake designs were considered; one being a solid 'door', and the second with holes inserted (see Figure 5.11). One of the concerns about the aerodynamic behaviour of the air brake system was type of wake that would be generated. In particular, there

were concerns about how this wake would interact with the complex rear wheel and suspension system downstream. It was important to ensure that the dominant vortex shedding frequencies from the air brake were well above the natural frequency of the rear wheel system (approx. 10Hz). This requirement led to the air brake design with holes which was an attempt to reduce the size and increase the frequency of the shed vortices. The flow visualisation approach set out in this paper has been a significant step towards a better understanding of the complex underlying flow behaviour in the wake downstream of the air brake. It has helped refine the position of the holes in the air brake door to optimise the behaviour of the air brake and minimise its impact downstream.

Overall, the benefits of the approach proposed here are: a speed up of the visualisation component of post processing CFD data for large datasets encountered in industrial CFD applications and thus a speed up of the overall engineering design cycle, consistency when comparing engineering designs using flow visualisation, and the automatic detection of otherwise undetectable (or difficult to detect) flow phenomena.

CHAPTER 6

Design of a Flow Visualisation Framework

COMPUTATIONAL fluid dynamics (CFD) is a rapidly developing tool for fluid analysis [ANS]. With wide reaching industrial applications such as automotive, aerospace, weather, and medical, it is an increasingly important software application used to speed up the design process while reducing costs. With increasing data complexity and size, the ability of the user to interpret the CFD results becomes increasingly difficult. Post processing software aims to alleviate this challenge by providing visualisation methods for the evaluation and correlation of simulation results. Bridging the gap between current visualisation software techniques and the latest research techniques is an important goal of our work.

Illustrated in Figure 6.1 is the CFD simulation pipeline which highlights the inputs, outputs, and processing steps. This process is described in three stages:

1. Preprocessing: A surface and the volumetric mesh is generated to model a physical object. This mesh is generated from computer aided design (CAD) geometry utilising mesh generation and manipulation tools. Boundary conditions and fluid properties are defined and specified.

2. Simulation: A computational simulation of the fluid is performed using numerical methods applied to the mesh, with respect to the boundary conditions and fluid properties.

3. Post Processing: The simulation result is explored, analysed, and visualised using a range of techniques dependent on the requirements of the CFD engineer.

In order to present a visualisation toolkit which is capable of dealing with large simulation data and complex algorithms, a comprehensive and versatile visualisation

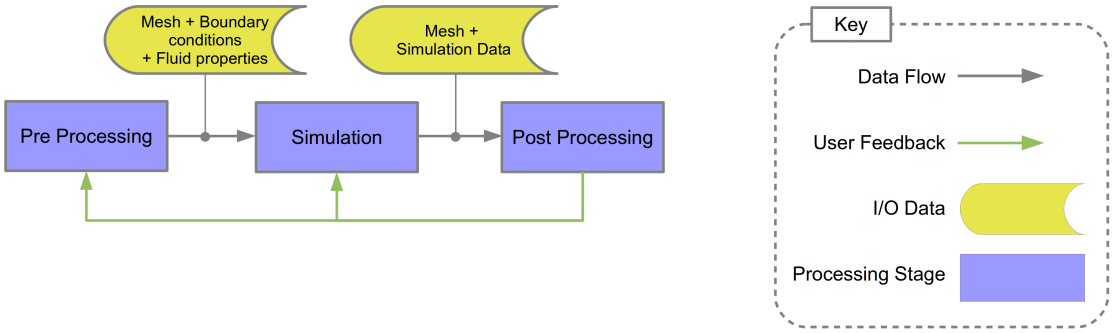


Figure 6.1: *The CFD simulation pipeline is comprised of a preprocessing stage, the simulation stage, and the post processing stage. This illustration shows the inputs and outputs of each stage, and highlights where the different tasks reside.*

framework is needed. It is the focus of this chapter to describe the design and implementation of a flow visualisation framework which provides scientists and engineers with effective solutions for the visualisation of CFD simulation data. By drawing on recent developments in the game and hardware industry, and by combining several scientific visualisation techniques our visualisation framework yields following benefits:

- A platform independent flexible framework which implements state of the art research algorithms in flow visualisation.
- A framework specifically designed to enable quick integration of new algorithms for study, testing, and evaluation.
- A smooth and efficient threaded user interface, even when processing large data and complex algorithms.
- Simultaneous comparisons of different techniques.

The remainder of this paper is organised as follows: Section 2 outlines the software application framework. Section 3 details the design of the GUI. Section 4 describes the services subsystem. Section 5 details the logic subsystem.

2 System Overview

In this section we start by discussing some design and implementation related work, we then provide an overview of our application framework and related subsystems. Laramee et al. [LHH05] describe the design and implementation of a flow visualisation subsystem which utilises the geometric and texture-based flow visualisation techniques.

In order to guarantee the quick responsiveness for user interaction even when dealing with large data, Piringer et al. [PTMB09] present a generic multi-threading architecture

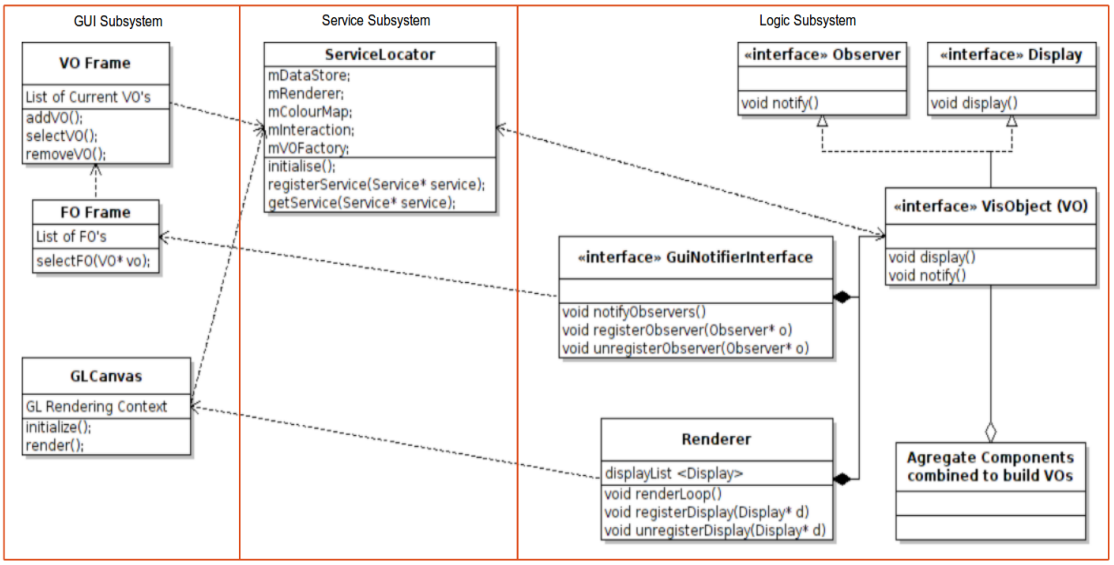


Figure 6.2: *This top level overview of the system shows three main subsections of the software application framework. The GUI subsystem provides the user interface and all associated user interaction and feedback. The logic subsystem encapsulates the processing aspects of the framework. The Services subsystem provides system wide services for use by the other subsystems.*

which enables early cancellation of the visualisation thread due to user interaction without common pitfalls of multi-threading. They also present an interactive visualisation toolkit, HyperMoVal [PBK10], as an implementation of this architecture in practice.

McLoughlin and Laramee [ML12] describe a flow visualisation software framework that offers a rich set of features. Their application also serves as a basis for the implementation and evaluation of new algorithms. The application is easily extendible and provides a clean interface for the addition of new modules. Peng et al. [PGL13] present a multi-linked framework which provides customised visualisation techniques for engineers to gain a fast overview and intuitive insight into the flow past the marine turbine.

Our software application system is divided into three main subsystems; the GUI, the Logic, and the Services. See Figure 6.2. We divide the system into smaller easier to maintain subsystems using logical groupings of similar functionality. The GUI subsystem is responsible for capturing the user input for computing a visualisation, displaying the resultant visualisation, and supporting user interaction with the visualisation. The Logic subsystem encapsulates the algorithms used to compute the visualisations. This subsystem processes the input data as specified by the user and computes the visualisation based on the given user input parameters and selected algorithm. Providing a central point of storage for input and derived data is the Service subsystem.

Another feature of our framework is the Visualisation Object (VO). The VO is an encapsulating aggregate of all the elements of an individual feature or individual algorithm.

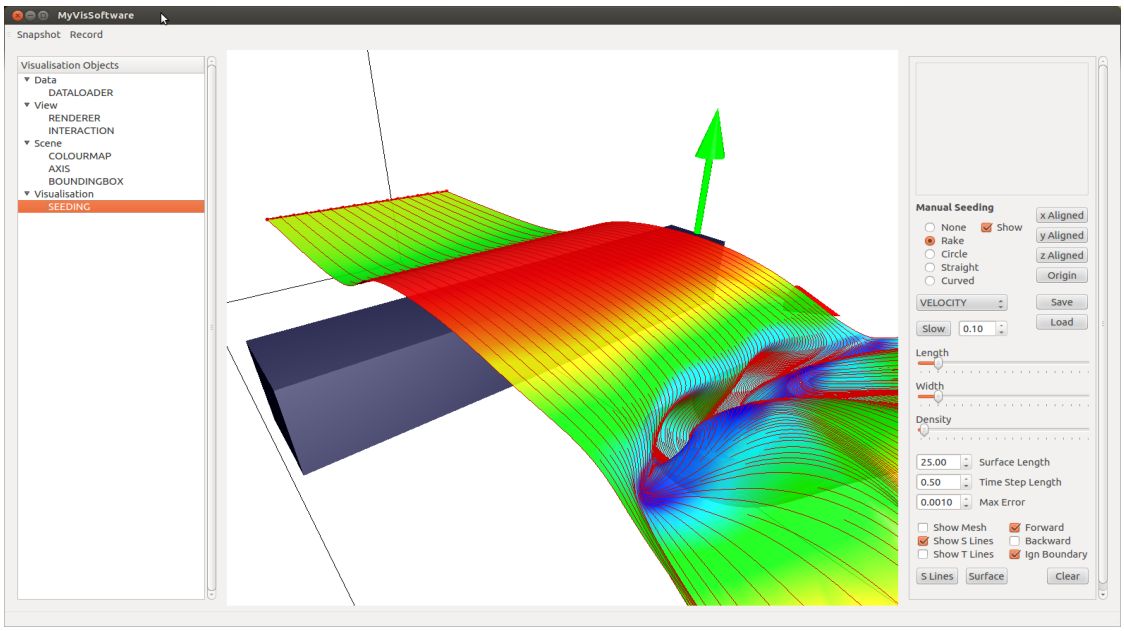


Figure 6.3: *This illustration shows the layout of the user interface.*

Some examples of features are: Interaction VO's which encapsulate the interaction with the visualisation e.g. panning, zooming, and rotating. Colour map VO's encapsulating colour map functionality. Stream surface VO's which encapsulates the stream surface generation algorithms [ELM*12].

For each VO within our framework there is an associated Frame Object (FO); an interface for encapsulating the I/O with the VO. This design approach creates a simple interface for adding new functionality quickly and easily. We further discuss VO's and FO's within our application, along with the detailed description of the structure of our framework in Sections 3, 5, and 4.

3 GUI Subsystem Design

The GUI is divided into three main frames. The centre frame for rendering the current list of VO's, the left frame for listing the current VO's, and the right frame area which displays the FO for the currently selected VO and lists the VO's currently available options. Refer to Figure 6.3.

VO's are added by right mouse clicking the parent VO category and selecting the required VO from the drop down menu. The VO categories (data, view, scene, visualisation) can be seen in the left frame of Figure 6.3. Right mouse clicking a VO in the list enables the user to delete it. Left mouse click the listed VO and its associated FO is displayed in the right frame area. The FO lists the attributes of the VO which can be manipulated. In Figure 6.3 the selected VO is a seeding object. The right frame displays

the the options available to the seeding object. At the top of this FO a track pad is available for seeding curve manipulation, under this we can specify type and size. We can save and load previously specified seeding curves. At the bottom of this FO we have options for the generation of stream surfaces.

Once we add the required VO's and set FO options, we can see the results rendered in the centre frame. Interaction with the visualisation is achieved using the mouse. For example: to pan press hold and drag the right mouse button, to zoom press hold and drag the left mouse button, to rotate the view press hold and drag the left and right mouse buttons together.

Each of the three frames within the interface run their actions in separate threads. This allows continued interaction with the interface (such as panning and rotation of the visualisation) while waiting for lengthy processes and computations (such as loading large datasets) to finish. As a result of our design approach the application of thread locks to specific VO's is trivial [cpp].

4 Services Subsystem Design

Some objects in an application can end up communicating with almost all other objects within the code base. It's difficult to find a component of our application that does not need access to objects which encapsulate either input or derived data at some point. Other examples are objects which encapsulate colour maps or OpenGL rendering. A method of accessing these objects via interfaces is also desirable. The interface can allow swapping of the current object for an alternative, or allow intermediate objects to be installed which can perform additional tasks such as logging. Systems like these can be thought of as services that need to be available to the entire application.

The Service Locator design pattern [Blac, Cod, Mar] is used to meet these requirements in our application. The Service Locator pattern decouples code that needs a service from both who it is (the concrete implementation type) and where it is (how we get to the instance of it). The Service subsystem utilises the Service Locator design pattern to provide a robust, type safe, central location for accessing system services.

In our services subsystem we focus on data storage and accessibility. The services we implement includes; raw data storage, derived data storage, model data storage, render object storage, colour map storage, interaction matrix storage, and a VO factory.

In Listing 6.1 we see the basic layout of the service locator class. The listing shows an initialisation function, a register function, and a get function. The service locator is reliant on a service being registered with it before any other objects try to use the service. The issue with this is that a null pointer could be returned if not initialised or a service is registered. It is also important to note that the service isn't aware of the concrete service class just the service interface. The only location in the code that knows about the concrete class is the function that registers the service.

If we try to use the service before a provider has been registered it returns a null

pointer. If the calling code doesn't check this we will crash the application. To address the null pointer issue we implement the Null Object design pattern [Blab] or in our case a null service. A null service inherits the same service interface but its implementation does nothing. This allows code that receives the object to safely continue on. The calling code will never know that the service isn't available, and we don't have to handle a null pointer. We are guaranteed a valid object.

Listing 6.1: *Service Locator Pattern*

```
#ifndef SERVICELOCATOR_H_
#define SERVICELOCATOR_H_

class ServiceLocator {
public:
	static void InitializeDataStore() {
		mDataStore = &mNullData;
	}

	static void Register(DataStoreInterface* service){
		if (service == nullptr) {
			mDataStore = &mNullData;
		} else {
			mDataStore = service;
		}
	}

	static DataStoreInterface& GetDataStore() {
		return *mDataStore;
	}

private:
	static DataStoreInterface* mDataStore;
	static NullDataStore mNullData;
}

#endif /* SERVICELOCATOR_H_ */
```

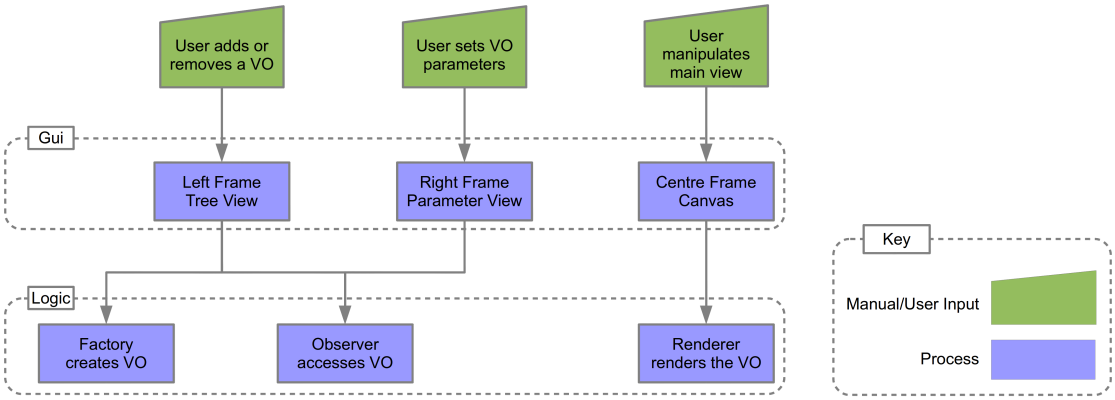


Figure 6.4: *Communication routes from the user to the logic subsystem. User interaction with the GUI is communicated to the visualisation object (VO) responsible for the selected visualisation algorithm via dedicated subsets of the logic subsystem.*

Our service locator is initialised within the main function prior to any other calls. The initialisation function registers the null objects. We can then register the any additional services as required. Registering a replacement service is a simple case of just registering the service. We don't delete any services within the service locator class, which provides us the flexibility of swapping services back and forth if required. An example of this is two interaction services storing different view matrices; which ever service is currently registered dictates from which viewpoint the visualisation is viewed in the centre frame.

Another important feature of this design pattern is the logging or debugging capabilities. Using a decorator design pattern [Blaa] we can easily add an intermediate object which can output either logging or debugging information. Accessing the services is a simple matter of calling the accessor function. This returns a reference to the currently registered service object. Unregistering a service can be achieved by registering a null pointer. The service locator will then register the default null service.

5 Logic Subsystem Design

The Logic subsystem encapsulates the algorithms used to compute the visualisations. This subsystem processes the input parameters selected by the user, and computes the visualisation based on the selected algorithm. Figure 6.4 shows the communication route from user interaction with the GUI, to the selected VO generating the visualisation data. There are three main subsets of the logic subsystem responsible for manipulation of the VO. These are; the factory class, the observer class and the renderer class. In this section a discussion of the Factory and Observer design patterns takes place. This is followed by a discussion of the render loop, and finishes with an examination of the aggregate VO and its life cycle.

The Factory

There are many cases when a class needs to create several types of object. This class can have this responsibility but it has to know about each and every object type. The possibility that the class cannot anticipate all the required objects in advance is clear. The idea of the factory pattern is to encapsulate the creation of a multitude of different classes utilising a common interface. A factory is the location of a class in the code where multiple types of object are constructed. The purpose of utilising this pattern is to isolate the creation of objects from their usage, and to create groups of similar objects without the need to depend on their concrete classes. This has the advantage of introducing new derived types with no change to the code that leverage the base class type. Using this pattern also introduces the possibility of interchanging the concrete implementations without code modification. An example of the factory pattern can be seen in Listing 6.2. If the user chooses to add a new VO in the treeview, the GUI instructs the factory to create a new VO of a type indicated by a type ID. A reference to the factory is stored in, and accessed from the Service Locator.

Listing 6.2: *Factory Pattern*

```
#ifndef COMPONENTFACTORY_H_
#define COMPONENTFACTORY_H_

class ComponentFactory {
public:
      ComponentFactory();
      ~ComponentFactory();

      VisObjectInterface* genObject(const VisTypeEnum& ID) {

            switch(ID) {
                  case VisTypeEnum::surfacevisobject:
                        return new SurfaceVisObject();
                        break;
                  case VisTypeEnum::axisvisobject:
                        return new AxisVisObject();
                        break;
                  default:
                        return new DefaultVisObject();
            }

      }
};

#endif /* COMPONENTFACTORY_H_ */
```

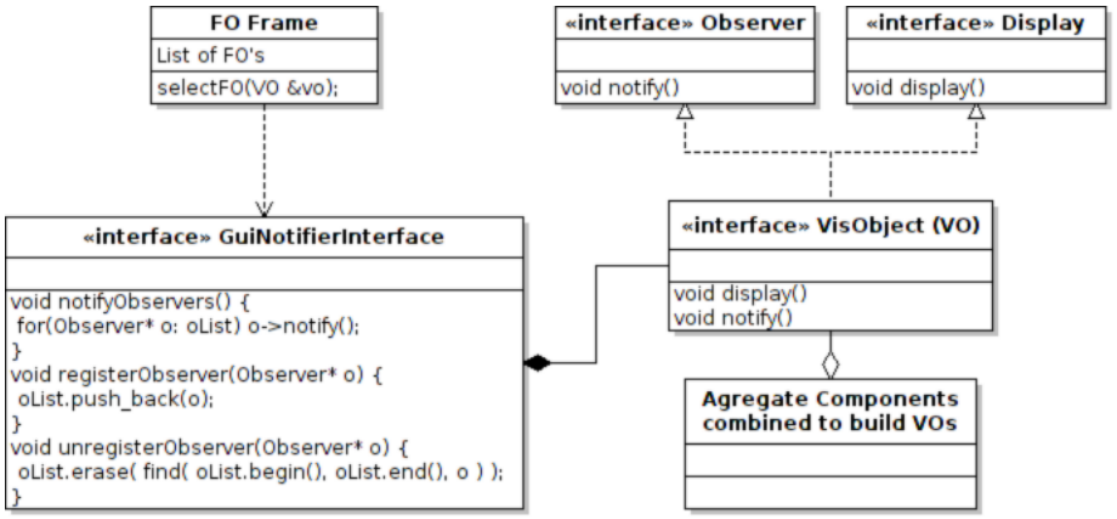


Figure 6.5: *This UML diagram shows the layout of the observer design pattern utilised for communication between the GUI frame objects (FO's) and the visualisation objects (VO's).*

The Observer

The observer design pattern is used to communicate an objects change of state to all dependant objects. These dependant objects are then automatically updated. This pattern controls the communication between classes. An object known as the subject is able to publish a change of state. A set of observer objects which depend on the subject are registered with the subject such that they can then be automatically notified when the state of the subject changes. This design pattern provides a loose coupling between the subject and its observers. The subject has access to a set of observers that are registered at runtime. The observers must inherit an observer interface thus the base class and its functionality is not known to, or required by the subject.

Figure 6.5 shows the the use case in our software framework. The VO's inherit the observer interface which includes the notify function which must be appropriately overridden in the base class. The VO is registered with the subject, which in our case is the GuiNotifier class. When a frame object (FO) state changes, it calls the notifyObservers function which notifies all registered VO's that the state has changed. The base class implementation overriding the notify function then checks for changes and updates itself accordingly where applicable.

The Rendering Loop

The GUI centre frame renders the current OpenGL rendering context to its canvas for viewing by the user 3. An example of this can be seen in the following illustration 6.3. In Figure 6.6 the centre frame encapsulates the GLCanvas class. When the GLCanvas

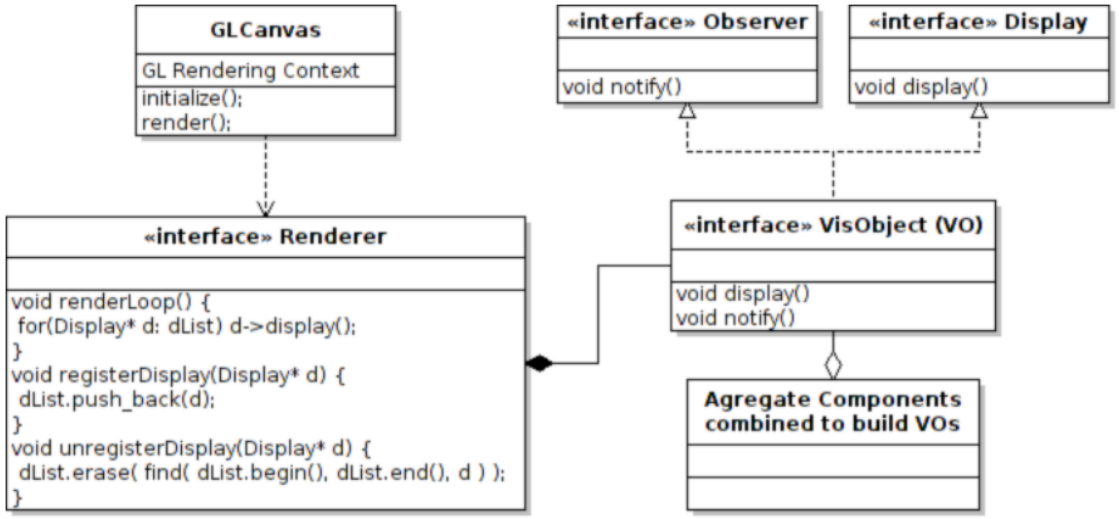


Figure 6.6: *This UML diagram shows the relationship between the renderer and the visualisation objects (VO's). Note the similarity with the observer design pattern shown in Figure 6.5*

class render event is triggered it calls the renderLoop function from the Renderer class.

The renderer class is designed to encapsulate the collection of OpenGL calls required to set the state of the OpenGL state machine. This class also has functionality based on the observer design pattern. VO objects are registered with the renderer, and when the renderLoop function is called by the GLCanvas class, the overridden display function called on the list of registered VO's. The detail of this is shown in the UML diagramme of Figure 6.6.

The general state of the scene is set from the renderer's associated frame object (FO). This includes features such as; perspective parameters, light source locations, lighting intensity's e.g. ambient, diffuse, and specular. The type of OpenGL Shader is also selected from this frame object. The renderer class defines how to render the scene data, whereas the VO defines what data is to be rendered.

The Renderer interface is used to provide a general interface for the specific OpenGL implementation. There are differing versions of OpenGL available supported by different specifications of hardware. The advantage of our system is the ability to interchange the Renderer class depending on the hardware support for a particular version of OpenGL. The required version of the Renderer class is registered with the Service Locator. This is also useful for testing revised OpenGL or GLSL [Opea] implementations.

What is GLSL: GLSL or Graphics Library Shading Language is a high level shading language which uses a syntax close to that of the C programming language. It was created for the purpose of providing developers with more precise control over the built in graphics pipeline. Previously developers would be required to use OpenGL assembly or

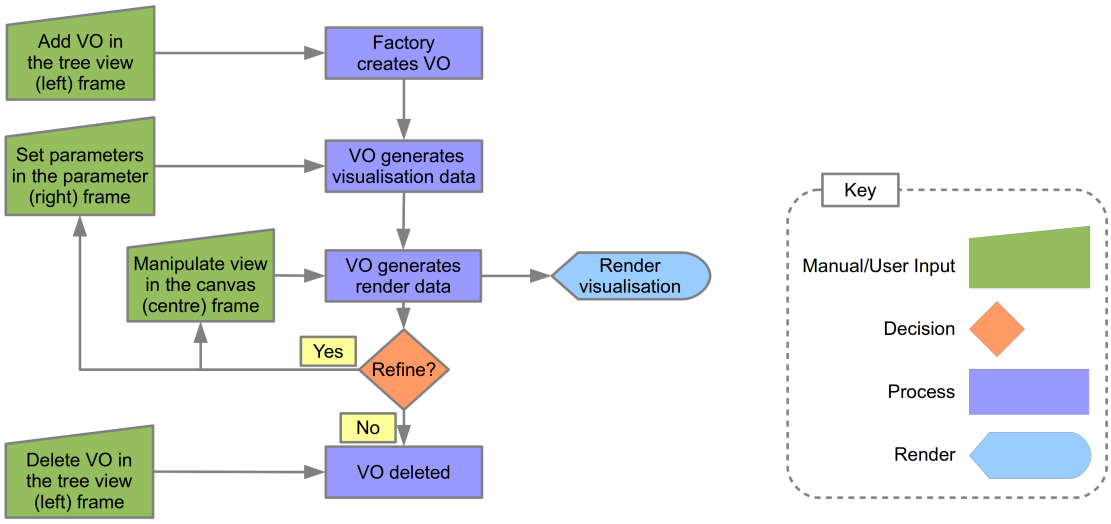


Figure 6.7: *The Visualisation Object (VO) lifecycle.*

graphics card specific programming languages to implement modifications to the fixed functionality pipeline.

In our implementation we use a combination of per pixel Phong lighting, edge highlighting using a Gaussian filter to detect sudden changes in depth, and depth peeling for order independent transparency. These algorithms are discussed in more detail in Chapters 2, 3, and 4, and in the work by Born et. al. [BWF*10] and Hummel et al. [HGH*10].

The Visualisation Object (VO)

The visualisation object (VO) is a key element in the functionality of this software framework. The entire framework is built around the idea of a single object interface type used for all implemented features within the code base. In this subsection we describe the lifecycle of a typical VO. We then describe the aggregate style implementation of the VO's and discuss some advantages of this approach over more traditional methods in the context of a robust visualisation framework for prototyping, testing, and comparing new techniques.

The life cycle of a VO is illustrated in Figure 6.7. The life cycle of the VO starts with its creation as the result of user input. The factory is responsible for the generation of VO's which are then passed to the GuiNotifierInterface detailed in Figure 6.5. Communication with the VO can then take place. The next step is setting the parameters of the VO required as input to the particular algorithm the VO encapsulates. Once these parameters are set the visualisation data is then generated.

During the visualisation generation process, data external to the VO may be required. An example of this is the vector and scalar data made available by another VO (the data

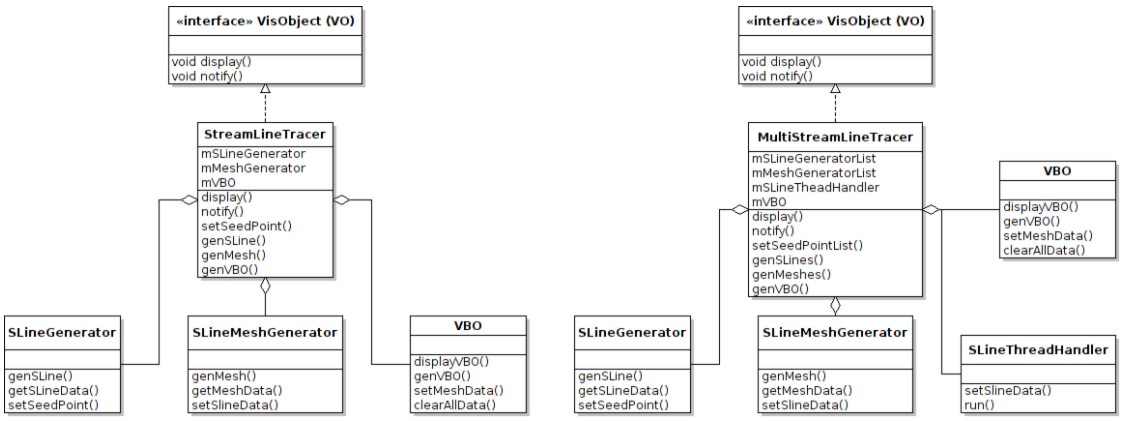


Figure 6.8: *This UML diagram illustrates a simplified version of a streamline tracer class and a multi threaded streamline tracer class. Rather than inherit the original streamline class and adding the multi threading implementation into the new class, we instead assemble a new class from the original components plus a new thread handling component. If threading was an original intention then the hierarchical inheritance could be designed more efficiently, but now we must refactor the code. With the component-based approach this foresight is not required. This is particularly useful in a system which is subject to adding a lot of new functionality.*

storage VO) registered with the service locator. This data can be accessed via the service locator interface, along with other data such as colour maps etc.

Once the visualisation data is generated, in the form of a triangulated mesh, the next step is to generate the render data. The mesh data is usually generated in dual arrays, one representing the vertex, normal, colour data, and one representing the node indices. This data is sent directly to the VBO class encapsulating the OpenGL Vertex Buffer Object. This class registers the pointers to the data, then specifies the required rendering parameters needed by the OpenGL state machine to correctly render the mesh.

Once the visualisation algorithm is rendered to the centre frame, the user is able to interact with the visualisation. The user can directly pan, zoom, and rotate the rendered scene by dragging the mouse over the scene in combination with the mouse buttons. The right mouse button is pressed to pan the scene, the left mouse button is pressed to zoom the scene, and both mouse buttons are used to rotate the scene. While interacting with the visualisation, the user can also set new parameters for the visualisation and generate a modified visualisation which is then rendered to screen. See Figure 6.7.

The final part of the process is deletion of the VO. If the user removes the VO from the left frame 6.3 by selecting delete from the VO context menu, the VO is then passed back to the factory for deletion ending this VO's lifecycle. All VO objects are created and deleted by the factory. This centralised approach makes for easy debugging of erroneous VO's during the VO lifecycle.

The VO's are constructed by aggregating the set of subclasses needed to compute the

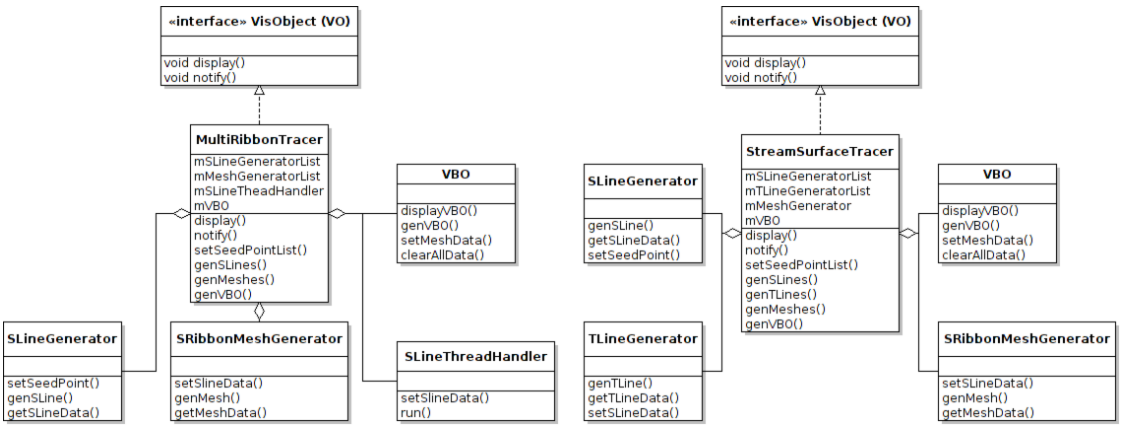


Figure 6.9: *This UML diagram illustrates a simplified version of a multi threaded stream ribbon tracer class and a stream surface tracer class. Some components are reused, and new components are added to provide the required functionality. In these illustrations the thread handler used in the stream ribbon tracer class is dropped from the stream surface tracer class, while the time line generator class is added for timeline refinement.*

required algorithm. The more traditional method of class hierarchies implement features available in a set of classes by extending them and therefore maximising code reuse. The design of the class hierarchy is made in advance with anticipation of possible additions to the code at a later date. This approach is tried and tested and generally works well.

However, a more recent approach is being more commonly utilised particularly within the games industries; the aggregate design pattern [Nys] [Wes]. With the more traditional hierarchical inheritance approach problems can arise in a number of ways. If the insight into the type of additions to the inheritance hierarchy were not well predicted, the code base can easily end up struggling to cope with the required changes leading to major refactoring of the code base. Also with more and more features being added the interwoven complexity of the inheritance hierarchical tree can quickly become unwieldy, and difficult to maintain. These scenarios do not lend themselves to quick and simple implementations of new features.

The aggregate design pattern provides an alternative approach which is more applicable to the type of framework discussed in this Chapter. We need to be able to add new algorithms quickly and easily but without the issues and complexity associated with large inheritance hierarchies. Another advantage of this approach is the ability to minimise the inherited overhead of unrequired features when extending a class of which only some of the functionality is required.

The parent class is constructed from an aggregation (collection) of a set of components classes. The component classes separate all the functionality required by the system into individual components which are independent of each other. The parent class becomes a wrapper for communication between each of the individual components i.e. the parent class performs no computation or stores any data with the exception

of the aggregated component classes, see Figures 6.8 and 6.9. Distinct new functionality is implemented in the system by creating a new component. New combinations of component classes are aggregated together to form new base classes providing the required system functionality.

Summary This chapter presents a comprehensive and versatile cross platform state of the art flow visualisation toolkit. This framework has the flexibility to add new algorithms fast and efficiently for testing, evaluation, comparing results, and for study by the domain experts. This chapter focusses on describing the design and implementation of a generic visualisation framework which provides scientists and engineers with effective solutions for the visualisation of CFD simulation data.

CHAPTER 7

Conclusions and Future Work

THE goal of this thesis is to study and propose effective methods for the seeding of stream surfaces. In order to achieve this objective this thesis presents several novel algorithms which address the challenges of placing stream surfaces at locations effectively capturing the underlying characteristics of the flow field. This work is completed in close collaboration with researchers from the Engineering CFD domain, and apply the proposed techniques to their data producing insightful visualisations for further analysis.

Chapter 2 surveys the current state of the art of flow visualisation with stream surfaces. These algorithms are discussed and categorised providing a concise overview of related work. Future work paths are discussed and provide the basis for further examination in this thesis. Following from this Chapter 3 focuses on placement techniques which fill the domain and require filtering. These algorithms are applied to data with differing flow characteristics testing the robustness of the approaches. Chapters 4 and 5 focus on placing stream surfaces adjacent to geometric flow structures. These approaches further address the challenges of clutter, visual perception, computational speed and memory footprint. The proposed algorithms and visualisations are evaluated by the domain experts.

This thesis is concluded first by restating the main contributions of each chapter, listed next in bullet form. This is followed by a detailed discussion of conclusions in Section 2. Then possible future work directions are presented in Section 3:

- Chapter 2 surveys the latest research developments in the area of flow visualisation with a focus on visualisation using surfaces, forming a concise overview of the related literature. Introduced is a classification scheme guided by Research

challenges including: construction, rendering, occlusion, and perception. This approach produces an intuitive grouping of papers that are naturally related to one another. The classification scheme highlights both mature areas of work where many solutions have been provided, and areas of work where unsolved problems remain. The study of literature focusing on flow visualisation with surfaces aided the formulation of new hypothesis and experiments which this thesis studies further.

- Chapter 3 describes an automatic approach to placing stream surfaces in 3D flow fields. The technique generates seeding curves from isolines derived from a scalar field at the domain boundary. The scalar field is derived from the angle of flow exit trajectory relative to the domain boundary plane. This chapter then discusses extending the concept to place surfaces throughout the domain rather than being confined to the domain boundary. After propagating surfaces from the domain boundary the algorithm then searches for free space on both sides of each surface. When free space is found a seeding curve is placed and used to trace a new steam surface in upstream and downstream directions. The free space search utilises a distance field for inter surface awareness and surface termination. Included are new techniques for reducing occlusion related to seeding multiple surfaces. Chapter 3 illustrates how to achieve adequate coverage of the domain capturing the features of the flow. Also introduced are improved techniques for surface filtering and perceptual enhancements such as interactive pixel filtering and clipping. The focus pays particular attention to seeding curve generation and occlusion.

- Chapter 4 presents an adaptation of a vector field clustering algorithm that can be guided by the user to automatically place stream surfaces. The clustering technique, driven by a distance measure based on error, is used to determine potential stream surface placement locations. Introduced are seeding curves generated by integrating through a derived curvature field. The seeding curves follow the curved flow structures maintaining orthogonality with the flow. Flow curvature, view dependant transparency, and view dependant saturation are combined with depth peeling and silhouette edge highlighting to provide enhancements to the perception of the visualisations. The results illustrate how to capture the characteristic structures within the flow field. The focus of this work pays particular attention to the flexibility of the clustering technique combined with the seeding curve generation strategies.

- Chapter 5 develops the concept presented in Chapter 4 closely working with the domain expert CFD engineers. The algorithm is tailored to work with large unstructured data fast and efficiently. The algorithm partitions the domain using k-means clustering combined with a novel distance function based on; a curvature field, a velocity gradient field, and normalised Euclidean distance. The focus pays particular attention to the performance, memory footprint, and flexibility of

the clustering. The seeding curve generation is modified for use with the k-means clustering. The work is evaluated by the domain expert CFD engineers.

- Chapter 6 presents the output from collaborations with domain experts CFD engineers in the form of a visualisation software framework. The framework unifies the presented algorithms into a cohesive application for continued use by the CFD engineers. We utilise advancements in game design technology, and leveraging features available with recent graphics hardware. We describe the design of our flow visualisation software framework, and discuss the effectiveness and scalability of the approach.

2 Conclusions

Despite the great amount of progress that has been made in the field of surface-based flow visualisation over the last two decades, a number of challenges remain. Challenges such as surface placement, speed of computation, memory footprint, perception, and evaluation remain key topics for further research. Chapter 2 provides a study of a variety of surface techniques and approaches for studying 3D vector field simulations. A summary and discussion of the challenges highlights both the unsolved problems and the mature areas where many solutions have been provided. There are some topics of study which could potentially benefit from further examination, experiment, and verification. Chapter 2 provides an up to date overview of the current state of the art providing an accurate guide for further research on the topic of flow visualisation with surfaces.

Chapter 3 studies the hypothesis that successful placement of stream surfaces could be achieved from the domain boundary when considering the boundary flow trajectory. This combined with the hypothesis that placing surfaces with a suitable density across the domain would capture all the characteristics of the flow field. An investigation of a range of methods for improving perception are studied as this approach can lead to very cluttered visualisations. Surface termination using a distance field proved an effective approach for maintaining a minimum distance between surfaces alleviating clutter. Interactive pixel filtering and transparency implemented in the rendering loop is effective at further reducing the visual clutter and occlusion. The placement strategy employed removes the need for the user to conduct lengthy examinations of the flow fields using manual seed placement techniques.

The algorithm is tested on a variety of simulations ranging from simple to complex in terms of data size, and fluid flow complexity. The techniques show adequate domain coverage capturing the features within the flow field. However some challenges remain with these approaches. First, the placement of stream surfaces at the boundary can only occur when there is flow crossing the boundary. Without this there can not be any seeding. Second, manual interaction with the filtering techniques is required to produce the final visualisation, with different settings being required for different scenarios. Third,

the quantity of stream surfaces required to fill very large datasets would prevent real time interaction.

To address these challenges we hypothesise that stream surfaces must be placed such that they capture the important characteristics of the flow field with minimal quantity. Seeding curves must be positioned and oriented in the neighbourhood of geometric structures best representing the flow field. Illustration techniques must enhance the perception of surface structures supporting the roll of effective placement.

To simplify the flow field and present potential seeding locations adjacent to important flow structures, hierarchical binary vector field clustering is presented in Chapter 4. The algorithm automates the capturing and visualisation of important characteristics within the flow field as defined by the user. The user is able to guide the visualisations by specifying the feature centred or overview clustering parameters. Emphasis is placed on flexibility allowing the density of seeding curves and their associated stream surfaces to be controlled by the user. The novel adaptation of an existing clustering method provides greater flexibility over the formation of the clusters and their locations. This approach leads to automatically locating seeding positions near important structures within the domain.

Chapter 4 discusses a novel technique for locally orienting straight seeding curves in line with a derived curvature field. This is further enhanced by integrating the seeding curves through the curvature field thus following the curvature of the flow structures. Chapter 4 also describes illustrative techniques which provide enhancements to the perception of the visualisations. Silhouette edge highlighting, combined view dependant and curvature dependant transparency, and view dependant saturation techniques are utilised to improve perception, reduce the visual clutter and occlusion associated with rendering multiple overlapping surfaces.

Chapter 4 demonstrates the feature centred and overview default settings of this approach for each dataset. The domain experts conclude that a set of standard parameter combinations would be beneficial for normal use. As a result of the flexibility of this framework, standard settings can be designed for use according to the requirements of the engineers using our technique. This approach produced effective visualisations, however some challenges remain. The memory footprint of this approach does not allow very large unstructured data to be examined due to main memory constraints. The binary clustering process is also slow and is not easily extendible to unstructured tetrahedral data.

The goal of the work in Chapter 5 is to improve computational performance, memory footprint, and robustness of the clustering technique from Chapter 4. This technique tailors stream surface seeding for the Bloodhound SSC project [Nob] which is characterised by large unstructured CFD data. We improve the performance and memory usage with the application of k-means clustering. The distance function, which combines flow curvature, velocity gradient, and Euclidean distance, produces comparable results with the technique from Chapter 4. While providing an environment and tools for the domain engineer to visualise undesirable flow behaviour related to vehicle drag, the domain ex-

perts provide feedback and conclude that this technique is a significant improvement.

In Chapter 6 we demonstrate how the simple and intuitive layout of the GUI is effective for reducing the software related learning curve by the domain experts. The subdivision of the GUI, Service Locater, and Logic into logical subsystems not only simplifies the system, but also simplifies the communication between the different subsystems, and provides greater flexibility when accessing the raw data. The use of aggregation with the VO's proved very successful and flexible when adding new algorithms and features to the software stack.

3 Future Work

Chapter 2 surveys, discusses, and summarises potential future work directions for flow visualisation with surfaces. Challenges such as: visualising error and uncertainty, extending current techniques for unsteady flow, interactive construction of time surfaces, large unstructured time dependant CFD data, perceptual challenges, information content, and evaluation of flow visualisation techniques, are highlighted as areas for further research. In this thesis we study algorithms for improved speed of computation, memory footprint of stream surface seeding, and we discuss our techniques with the aid of domain expert feedback.

There are still a number of challenges remaining for the effective placement and illustration of integral surfaces within a 3D flow field. The proposed techniques studied in this thesis require input parameters to be specified by the user prior to generating the visualisations. In Chapter 3 the choice of iso value for isoline generation at the boundary, and the surface separation values for the interior surfaces, are used to guide the visualisation results. The illustrative techniques also require user input to produce the required results. In Chapters 4 and 5 user input is required for the distance functions again for guiding the resultant visualisations. Although improvements are made to the illustrative techniques user input is still required for the level of transparency. The domain experts are happy with the flexibility of guiding the visualisation results for differing cases, however the ultimate goal must be to eliminate the need for input parameters to be set. As it stands the user must understand the effect of changing the input parameters in advance of generating the visualisations. If this requirement is removed while maintaining the desired level of visualisation performance, it will simplify the users task and thus must be an important direction of future work.

The choice of the initial clusters in Chapter 5 impact the results achieved by k-means clustering after convergence to the local minimum. We do not guarantee the results are globally optimal. Also, the clustering results can produce cluster centres which may reside outside the flow domain e.g. inside the object of study as a result of a cluster boundary straddling the object in Euclidean space. A similar issue is present with the clustering technique in Chapter 4. *U-shaped* clusters can form as a result of the hierarchical binary clustering process. An in depth study of these challenges and their

effect on the results would be a valuable future contribution.

There is also potential to extend the algorithm in Chapter 5 to path and streak surface placement. An examination of the memory and performance requirements associated with extending the algorithm to time dependent flow would be required. The perception and information content of the surfaces, when modified to work with unsteady flow behaviour, would also be key topics for further research.

Visualising error or uncertainty introduced by the underlying numerical techniques, or from the sensitivity of the algorithms used to generate visualisations, could lead to misleading information. This points to possible future work developing techniques to visualise these different types of error and uncertainty. Another potential future work direction is the study of multivariate CFD data. Often CFD data contains not only vector data, but also a range of additionally computed attributes such as pressure, density, temperature, turbulence intensity etc. Utilisation of these attributes to compute meaningful visualisations must be an important goal of future work.

Future work for the visualisation framework described in Chapter 6 includes further abstraction of the VO's and FO's to a plugin-based system where new features and algorithms can by added via a plugin interface. This would further simplify implementation and remove the need to modify the main code base. Another area of further work is extending the aggregate pattern of composite components to an XML-based approach for constructing the VO's. This combined with a plugin interface for the components would produce a very flexible way to add new features and algorithms to the software.

Another topic of future work for our visualisation framework is the use of a rainbow colour map. Although not studied within this thesis, research has shown that the rainbow colour map is rarely the optimal choice when displaying data. This particular colour map may confuse the user due to; reduced perceptual ordering, uncontrolled luminance variation, and the introduction of non data-dependent gradients [BTI07]. The application of more suitable colour maps within our software framework, with consideration for issues related to colour blending while rendering transparency, must therefore be an important next step.

Bibliography

[Abo] ABORIGINAL ART ONLINE: Rock Art. http://www.aboriginalartonline.com/art/rock.php. Accessed: May 2013. 2

[And11] ANDERSON JR J. D.: *Fundamentals of Aerodynamics*, fifth edition with si units ed. McGraw-Hill Higher Education, 2011. 5, 6

[ANS] ANSYS UK: Fluent Engineering Simulation. http://www.fluent.co.uk/. Accessed: April 2013. 5, 109

[AS92] ABRAHAM R. H., SHAW C. D.: *Dynamics - the Geometry of Behavior*. Addison-Wesley, 1992. 39

[Bel87] BELIE R. G.: Some Advances in Digital Flow Visualization. In *AIAA Aerospace Sciences Conference* (Reno, NV, January 1987), AIAA, pp. 87–1179. 24

[BFTW09] BUERGER K., FERSTL F., THEISEL H., WESTERMANN R.: Interactive Streak Surface Visualization on the GPU. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1259–1266. xv, 17, 31, 46, 47

[Blaa] BLACKWASP: Decorator Design Pattern. http://www.blackwasp.co.uk/Decorator.aspx. Accessed: May 2013. 115

[Blab] BLACKWASP: Null Object Design Pattern. http://www.blackwasp.co.uk/NullObject.aspx. Accessed: May 2013. 114

[Blac] BLACKWASP: Service Locator Design Pattern. http://www.blackwasp.co.uk/ServiceLocator.aspx. Accessed: May 2013. 113

[BM08] BAVOLI L., MYERS K.: Order Independent Transparency with Dual Depth Peeling. *NVIDIA Developer SDK 10* (February 2008). 79

[Boc07] BOCK H.-H.: Clustering Methods: A History of k-Means Algorithms. In *Selected Contributions in Data Analysis and Classification*, Brito P., Cucumel G., Bertrand P., Carvalho F., (Eds.), Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg, 2007, pp. 161–172. 88, 92

[Bri03] BRIDSON R. E.: *Computational Aspects of Dynamic Surfaces*. PhD thesis, Stanford, CA, USA, 2003. 30

[BSDW12] BACHTHALER S., SADLO F., DACHSBACHER C., WEISKOPF D.: Space-Time Visualization of Dynamics in Lagrangian Coherent Structures of Time-Dependent 2D Vector Fields. *International Conference on Information Visualization Theory and Applications* (2012), 573–583. 17, 35

[BSH97] BATTKE H., STALLING D., HEGE H.: Fast Line Integral Convolution for Arbitrary Surfaces in 3D. In *Visualization and Mathematics* (1997), Springer-Verlag, pp. 181–195. 17, 42

[BSWE06] BACHTHALER S., STRENGERT M., WEISKOPF D., ERTL T.: Parallel texture-based vector field visualization on curved surfaces using GPU cluster computers. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV06) , pages 75-82. Eurographics Association, 2006* (2006), Universität Stuttgart. 17, 44, 47

[BTI07] BORLAND D., TAYLOR II R. M.: Rainbow color map (still) considered harmful. *IEEE computer graphics and applications 27*, 2 (2007), 14–17. 128

[BW08] BACHTHALER S., WEISKOPF D.: Animation of Orthogonal Texture Patterns for Vector Field Visualization. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (July 2008), 741–755. 26

[BWF*10] BORN S., WIEBEL A., FRIEDRICH J., SCHEUERMANN G., BARTZ D.: Illustrative Stream Surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1329–1338. 17, 40, 47, 84, 100, 119

[CCK07] CHEN Y., COHEN J. D., KROLIK J.: Similarity-Guided Streamline Placement with Error Evaluation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1448–1455. 50

[CEI] CEI SOFTWARE: EnSight. http://www.ceisoftware.com/. Accessed: April 2013. 5

[CL93] CABRAL B., LEEDOM L. C.: Imaging Vector Fields Using Line Integral Convolution. In *Poceedings of ACM SIGGRAPH 1993* (1993), Annual Conference Series, pp. 263–272. 21, 41

[Cod] CODE PROJECT: A Basic Introduction On Service Locator Pattern. http://www.codeproject.com/Articles/18464/

A-Basic-Introduction-On-Service-Locator-Pattern. Accessed: May 2013. 113

[cpp] CPPREFERENCE.COM: std::lock. http://en.cppreference.com/w/cpp/thread/lock. Accessed: May 2013. 113

[CSBI05] CAMARRI S., SALVETTI M. V., BUFFONI M., IOLLO A.: Simulation of the Three-Dimensional Flow Around a Square Cylinder Between Parallel Walls at Moderate Reynolds Numbers. In *Congresso di Meccanica ed Applicata* (2005), pp. 11–15. xvi, xx, 60, 96

[CSFP12] CARNECKY R., SCHINDLER B., FUCHS R., PEIKERT R.: Multi-layer Illustrative Dense Flow Visualization. *Computer Graphics Forum 31*, 3 (2012), 895–904. 62

[DGS79] DOUGLAS J. F., GASIOREK J. M., SWAFFIELD J. A.: *Fluid Mechanics*, 1 ed. Pitman Publishing Limited, 1979. 85

[DH92] DARMOFAL D., HAIMES R.: *Visualization of 3-D Vector Fields: Variations on a Stream*. Paper 92-0074, AIAA, 1992. 25

[dLvW95] DE LEEUW W., VAN WIJK J. J.: Enhanced Spot Noise for Vector Field Visualization. In *Proceedings IEEE Visualization '95* (Oct. 1995), IEEE Computer Society, pp. 233–239. 17, 41

[EL07] EITZ M., LIXU G.: Hierarchical spatial hashing for real-time collision detection. In *Shape Modeling and Applications, 2007. SMI'07. IEEE International Conference on* (2007), IEEE, pp. 61–70. 88, 145

[EL13] EDMUNDS M., LARAMEE R. S.: *Design of a Flow Visualisation Framework*. Tech. rep., The Visual and Interactive Computing Group, Computer Science Department, Swansea University, Wales, UK, 2013. 14

[ELC*12a] EDMUNDS M., LARAMEE R., CHEN G., ZHANG E., MAX N.: Advanced, Automatic Stream Surface Seeding and Filtering. In *Theory and Practice of Computer Graphics* (2012), pp. 53–60. 13

[ELC*12b] EDMUNDS M., LARAMEE R. S., CHEN G., MAX N., ZHANG E., WARE C.: Surface-Based Flow Visualization. *Computers & Graphics 36*, 8 (2012), 974 – 990. 7, 12

[ELEC13] EDMUNDS M., LARAMEE R. S., EVANS B., CHEN G.: *Stream Surface Placement for a Land Speed Record Vehicle*. Tech. rep., The Visual and Interactive Computing Group, Computer Science Department, Swansea University, Wales, UK, 2013. 13

[ELM*12] EDMUNDS M., LARAMEE R., MALKI R., MASTERS I., CROFT T., CHEN G., ZHANG E.: Automatic Stream Surface Seeding: A Feature Centered Approach. *Computer Graphics Forum 31*, 3.2 (June 2012), 1095–1104. xx, 13, 95, 96, 112

[EML*11] EDMUNDS M., MCLOUGHLIN T., LARAMEE R. S., CHEN G., ZHANG E., MAX N.: Automatic Stream Surfaces Seeding. In *EUROGRAPHICS 2011 Short Papers* (Llandudno, Wales, UK, April 11–15 2011), pp. 53–56. 13

[FBTW10] FERSTL F., BURGER K., THEISEL H., WESTERMANN R.: Interactive Separating Streak Surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (November-December 2010), 1569–1577. xv, 17, 21, 32, 46

[FC95] FORSSELL L. K., COHEN S. D.: Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June 1995), 133–141. 17, 36, 41, 42

[FCL09] FORSBERG A. S., CHEN J., LAIDLAW D. H.: Comparing 3D Vector Field Visualization Methods: A User Study. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1219–1226. 48

[FKSS12] FERREIRA N., KLOSOWSKI J. T., SCHEIDEGGER C. E., SILVA C. T.: Vector field k-means: Clustering trajectories by fitting multiple vector fields. *CoRR abs/1208.5801* (2012). 95

[For94] FORSSELL L. K.: Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution. In *Proceedings IEEE Visualization '94* (Oct. 1994), IEEE Computer Society, pp. 240–247. 42

[Fri08] FRIENDLY M.: A brief history of data visualization. In *Handbook of data visualization*. Springer, 2008, pp. 15–56. xiii, 1, 2

[FW12] FANG Y., WANG J.: Selection of the number of clusters via the bootstrap method. *Computational Statistics and Data Analysis 56*, 3 (2012), 468 – 477. 95

[Gel01] GELDER A. V.: Stream Surface Generation for Fluid Flow Solutions on Curvilinear Grids. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualizatation (VisSym-01)* (May 28–30 2001), Ebert D., Favre J. M., Peikert R., (Eds.), Springer-Verlag, pp. 95–106. 17, 22, 34, 46

[GKT*08] GARTH C., KRISHNAN H., TRICOCHE X., TRICOCHE T., JOY K. I.: Generation of Accurate Integral Surfaces in Time-Dependent Vector Fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1404–1411. xiii, xiv, 3, 17, 23, 26, 27, 49, 55, 77, 97

[GOV] GOV: Met Office. http://www.metoffice.gov.uk/. Accessed: 08th August 2010. 4

[GTS*04] GARTH C., TRICOCHE X., SALZBRUNN T., BOBACH T., SCHEUERMANN G.: Surface Techniques for Vortex Visualization. In *Joint Eurographics - IEEE TCVG Symposium on Visualization* (2004), Deussen O., Hansen C., Keim D., Saupe D., (Eds.), Eurographics Association, pp. 155–164. xiv, 17, 25, 28, 44, 47

[Hal01] HALLER G.: Distinguished Material Surfaces and Coherent Structures in Three Dimensional Fluid Flows. *Physica D 149* (2001), 248–277. 33

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K.: IRIS: Illustrative Rendering for Integral Surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1319–1328. xv, 17, 37, 40, 47, 84, 100, 119

[Hir89] HIRSCH C.: *Numerical Computation of Internal and External Flows*. Butterworth-Heinemann, 1989. 6

[Hom91] HOMICZ G. F.: *Numerical Simulation of VAWT Stochastic Aerodynamic Loads Produced by Atmospheric Turbulence: VAWT-SAL Code*. Sandia National Laboratories, 1991. 85

[Hos94] HOSEA M.E. AND SHAMPINE L.F.: Estimating the Error of the Classic Runge–Kutta Formula. *Applied Mathematics and Computation 66*, 2–3 (1994), 217 – 226. 146

[Hul92] HULTQUIST J. P. M.: Constructing Stream Surfaces in Steady 3D Vector Fields. In *Proceedings IEEE Visualization '92* (1992), pp. 171–178. 17, 18, 22, 24, 26, 34, 46

[IEE] IEEE VISWEEK: 2008 IEEE Visualization Design Contest. http://viscontest.sdsc.edu/2008/. Accessed: 08th August 2010. 4

[JBS06] JONES M. W., BAERENTZEN J. A., SRAMEK M.: 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics 12* (2006), 581–599. 56

[JEH01] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Lagrangian-Eulerian Advection for Unsteady Flow Visualization. In *Proceedings IEEE Visualization '01* (October 2001), IEEE Computer Society, pp. 53–60. 44

[JL97] JOBARD B., LEFER W.: Creating Evenly–Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing '97* (1997), vol. 7, pp. 45–55. 40, 50

[Ker90] KERLIC D. G.: Moving Iconic Objects in Scientific Visualization '91. In *Proceedings of Visualization* (October 1990), pp. 124–129. 24

[KGJ09] KRISHNAN H., GARTH C., JOY K.: Time and Streak Surfaces for Flow Visualization in Large Time-Varying Data Sets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1267–1274. xiv, 17, 22, 23, 30, 46

[Kog07] KOGAN J.: *Introduction to Clustering Large and High-Dimensional Data*. Cambridge University Press, 2007. 88, 92

[Lar02] LARAMEE R.: Interactive 3D Flow Visualization Using a Streamrunner. In *CHI 2002, Conference on Human Factors in Computing Systems, Extended Abstracts* (April 20–25 2002), ACM SIGCHI, ACM Press, pp. 804–805. 50

[Lar03] LARAMEE R. S.: FIRST: A Flexible and Interactive Resampling Tool for CFD Simulation Data. *Computers & Graphics 27*, 6 (2003), 905–916. 18

[Lar08] LARAMEE R. S.: Flow Visualization: The State-of-the-Art. In *The 19th Conference of Simulation and Visualization (SimVis)* (Magdeburg, Germany, February 28–29 2008). (Invited Talk). 7

[LCS*12] LIU Z., CAI S., SWAN II J. E., MOORHEAD II R. J., MARTIN J. P., JANKUN-KELLY T. J.: A 2D Flow Visualization User Study Using Explicit Flow Synthesis and Implicit Task Design. *IEEE Transactions on Visualization and Computer Graphics 18*, 5 (2012), 783–796. 48

[LEG*08] LARAMEE R. S., ERLEBACHER G., GARTH C., THEISEL H., TRICOCHE X., WEINKAUF T., WEISKOPF D.: Applications of Texture-Based Flow Visualization. *Engineering Applications of Computational Fluid Mechanics (EACFM) 2*, 3 (Sept. 2008), 264–274. xiii, 8, 15

[LGD*05] LARAMEE R. S., GARTH C., DOLEISCH H., SCHNEIDER J., HAUSER H., HAGEN H.: Visual Analysis and Exploration of Fluid Flow in a Cooling Jacket. In *Proceedings IEEE Visualization 2005* (2005), pp. 623–630. 17, 44, 50, 63

[LGSH06] LARAMEE R. S., GARTH C., SCHNEIDER J., HAUSER H.: Texture-Advection on Stream Surfaces: A Novel Hybrid Visualization Applied to CFD Results. In *Data Visualization, The Joint Eurographics-IEEE VGTC Symposium on Visualization (EuroVis 2006)* (2006), Eurographics Association, pp. 155–162,368. 17, 18, 44, 51

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., POST F. H., VROLIJK B., WEISKOPF D.: The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum 23*, 2 (June 2004), 203–221. 7, 15, 45

[LHH05] LARAMEE R. S., HADWIGER M., HAUSER H.: Design and Implementation of Geometric and Texture-Based Flow Visualization Techniques. In *Proceedings of the 21st Spring Conference on Computer Graphics* (May 2005), pp. 67–74. 110

[LHZP07] LARAMEE R., HAUSER H., ZHAO L., POST F. H.: Topology-Based Flow Visualization: The State of the Art. In *Topology-Based Methods in Visualization (Proceedings of Topo-in-Vis 2005)* (2007), Mathematics and Visualization, Springer, pp. 1–19. 7, 64

[LJH03] LARAMEE R., JOBARD B., HAUSER H.: Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proceedings IEEE Visualization '03* (2003), IEEE Computer Society, pp. 131–138. 17, 37, 43, 44

[LKJ*05] LAIDLAW D., KIRBY R., JACKSON C., DAVIDSON J., MILLER T., DA SILVA M., WARREN W., TARR M.: Comparing 2D Vector Field Visualization Methods: A User Study. *IEEE Transactions on Visualization and Computer Graphics 11*, 1 (jan 2005), 59–70. 48

[LMG97] LOFFELMANN H., MROZ L., GROLLER E.: *Hierarchical Streamarrows for the Visualization of Dynamical Systems*. Technical report, Institute of Computer Graphics, Vienna University of Technology, 1997. xv, 17, 39

[LMGP97] LOFFELMANN H., MROZ L., GROLLER E., PURGATHOFER W.: Stream Arrows: Enhancing the Use of Streamsurfaces for the Visualization of Dynamical Systems. *The Visual Computer 13* (1997), 359–369. 17, 18, 39, 51

[LS07] LI L., SHEN H.-W.: Image-Based Streamline Generation and Rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 630–640. 50

[LSH04] LARAMEE R. S., SCHNEIDER J., HAUSER H.: Texture-Based Flow Visualization on Isosurfaces from Computational Fluid Dynamics. In *Data Visualization, The Joint Eurographics-IEEE TVCG Symposium on Visualization (VisSym '04)* (2004), Eurographics Association, pp. 85–90,342. 17, 44

[LTWH08] LI G.-S., TRICOCHE X., WEISKOPF D., HANSEN C. D.: Flow Charts: Visualization of Vector Fields on Arbitrary Surfaces. *IEEE Transactions on Visualization and Computer Graphics 14*, 5 (2008), 1067–1080. 17, 45

[LvWJH04] LARAMEE R., VAN WIJK J. J., JOBARD B., HAUSER H.: ISA and IBFVS: Image Space Based Visualization of Flow on Surfaces. *IEEE Transactions on Visualization and Computer Graphics 10*, 6 (Nov. 2004), 637–648. xv, 17, 38, 43, 44

[LWSH04] LARAMEE R., WEISKOPF D., SCHNEIDER J., HAUSER H.: Investigating Swirl and Tumble Flow with a Comparison of Visualization Techniques. In *Proceedings IEEE Visualization 2004* (2004), pp. 51–58. 17, 44, 50

[Lyn] LYNN MCDONALD: Florence Nightingale A Hundred Years On. http://www.uoguelph.ca/~cwfn/short/whr.html. Accessed: May 2013. 2

[Mar] MARTIN FOWLER: Inversion of Control Containers and the Dependency Injection pattern. http://martinfowler.com/articles/injection.html. Accessed: May 2013. 113

[Mas99] MASE G. T.: *Continuum Mechanics for Engineers*. CRC Press, 1999. 33

[MCHM10] MARCHESIN S., CHEN C.-K., HO C., MA K.-L.: View-Dependent Streamlines for 3D Vector Fields. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010). 50

[Mer] MERNAGH, M.: Napoleon's Russian Campaign. http://www.significancemagazine.org/details/webexclusive/2038005/Napoleons-Russian-Campaign---200-years-on.html. Accessed: May 2013. 2

[MH02] MÖLLER T., HAINES E.: *Real-Time Rendering*, 2 ed. A. K. Peters Limited, 2002. 79, 98

[MKFI97] MAO X., KIKUKAWA M., FUJITA N., IMAMIYA A.: Line Integral Convolution for 3D Surfaces. In *Visualization in Scientific Computing '97. Proceedings of the Eurographics Workshop* (1997), Eurographics, pp. 57–70. 17, 42

[ML12] MCLOUGHLIN T., LARAMEE R. S.: *Design and Implementation of Interactive Flow Visualization Techniques*. InTech Publishers, 2012, ch. 6, pp. 87–110. 111

[MLP*10] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over Two Decades of Integration-Based, Geometric Flow Visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829. 7

[MLZ09] MCLOUGHLIN T., LARAMEE R. S., ZHANG E.: Easy Integral Surfaces: A Fast, Quad-Based Stream and Path Surface Algorithm. In *Proceedings of Computer Graphics International (CGI '09)* (May 2009), Computer Graphics Society, Springer, pp. 67–76. xiv, 17, 22, 24, 27, 46

[MLZ10] MCLOUGHLIN T., LARAMEE R. S., ZHANG E.: Constructing Streak Surfaces in 3D Unsteady Vector Fields. In *Proceedings Spring Conference on Computer Graphics* (Dec 2010), Hauser H., (Ed.), pp. 25–32. 17, 22, 32, 46

[MMWC11] MALKI R., MASTERS I., WILLIAMS A. J., CROFT T. N.: The Influence of Tidal Stream Turbine Spacing on Performance. In *Procedings of the 9th European Wave and Tidal Energy Conference (EWTEC)* (Southampton, England, UK, 2011). 85, 86

[MS93] MA K.-L., SMITH P. J.: Cloud Tracing in Convection-Diffusion Systems. In *VIS '93: Proceedings of the 4th conference on Visualization '93* (Washington, DC, USA, 1993), IEEE Computer Society, pp. 253–260. 25

[MT*03] MATTAUSCH O., THEUSSL T., , HAUSER H., GRÖLLER E.: Strategies for Interactive Exploration of 3D Flow Using Evenly-Spaced Illuminated Streamlines. In *Proceedings of the 19th Spring Conference on Computer Graphics* (2003), pp. 213–222. xiv, 9, 50

[MT04] MILGRAM R. J., TRINKLE J. C.: The geometry of configuration spaces for closed chains in two and three dimensions. *Homology, Homotopy and Applications 6*, 1 (2004), 237–267. 18

[MYO10] MUNSON B. R., YOUNG D. F., OKIISHI T. H.: *Fundamentals of Fluid Mechanics*, sixth edition with si units ed. John Wiley and Sons, 2010. 5, 6

[Nob] NOBLE, R.: Bloodhound SSC. http://www.bloodhoundssc.com/. Accessed: March 2013. 11, 13, 88, 126

[Nys] NYSTROM, R.: Game Programming Patterns. http://gameprogrammingpatterns.com/component.html. Accessed: July 2013. 121

[Opea] OPENGL.ORG: OpenGL GLSL. http://www.opengl.org/wiki/OpenGL_Shading_Language. Accessed: July 2013. 118

[Opeb] OPENGL.ORG: OpenGL Primitives. http://www.opengl.org/wiki/Primitive. Accessed: July 2013. 21

[Par82] PARASCHIVOIU I.: Aerodynamic loads and performance of the Darrieus Rotor. *J. Energy 6*, 6 (1982), 406–412. 85

[PBK10] PIRINGER H., BERGER W., KRASSER J.: HyperMoVal: Interactive Visual Validation of Regression Models for Real-Time Simulation. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 983–992. 111

[PCY09] PALMERIUS K. L., COOPER M., YNNERMAN A.: Flow Field Visualization Using Vector Field Perpendicular Surfaces. In *Spring Conference on Computer Graphics* (2009). 17, 27

[Pea85] PEACHEY D. R.: Solid Texturing of Complex Surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 85) 19*, 3 (1985), 279–286. 42

[PGL*12] PENG Z., GRUNDY E., LARAMEE R. S., CHEN G., CROFT N.: Mesh-Driven Vector Field Clustering and Visualization: An Image-Based Approach. *IEEE Transactions on Visualization and Computer Graphics 18*, 2 (2012), 283–298. xiii, xv, 8, 17, 38, 46, 64, 84, 89

[PGL13] PENG Z., GENG Z., LARAMEE R. S.: *Design and Implementation of a System for Interactive, High-Dimensional Vector Field Visualization*, vol. 8. Nova Science Publishers, 2013, pp. 175–200. 111

[PL08] PENG Z., LARAMEE R. S.: Vector Glyphs for Surfaces: A Fast and Simple Glyph Placement Algorithm for Adaptive Resolution Meshes. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2008* (2008), pp. 61–70. 17, 38, 46

[PL09] PENG Z., LARAMEE R. S.: Higher Dimensional Vector Field Visualization: A Survey. In *Theory and Practice of Computer Graphics (TPCG '09)* (June 2009), pp. 149–163. 7

[PNB02] POST F. H., NIELSON G. M., BONNEAU G. P. (Eds.): *Data Visualization: The State of the Art*. Springer, 2002. 3

[PPF*10] POBITZER A., PEIKERT R., FUCHS R., SCHINDLER B., KUHN A., THEISEL H., MATKOVIC K., HAUSER H.: On The Way Towards Topology-Based Visualization of Unsteady Flow the State Of The Art. *EuroGraphics 2010 State of the Art Reports* (2010), 137–154. 33

[PR99] PEIKERT R., ROTH M.: The Parallel Vectors Operator - A Vector Field Visualization Primitive. In *Proceedings of IEEE Visualization '99* (1999), IEEE Computer Society, pp. 263–270. 35

[PS09] PEIKERT R., SADLO F.: Topologically Relevant Stream Surfaces for Flow Visualization. In *Proc. Spring Conference on Computer Graphics* (April 2009), Hauser H., (Ed.), pp. 43–50. xiv, 17, 22, 28, 29, 47, 64

[PTMB09] PIRINGER H., TOMINSKI C., MUIGG P., BERGER W.: A Multi-threading Architecture to Support Interactive Visual Exploration. *Visualization and Computer Graphics, IEEE Transactions on 15*, 6 (2009), 1113–1120. 110

[PVH*03] POST F. H., VROLIJK B., HAUSER H., LARAMEE R. S., DOLEISCH H.: The State of the Art in Flow Visualization: Feature Extraction and Tracking. *Computer Graphics Forum 22*, 4 (Dec. 2003), 775–792. 7, 15, 63

[PW94] PAGENDARM H.-G., WALTER B.: Feature Detection from Vector Quantities in a Numerically Simulated Hypersonic Flow Field in combination with Experimental Flow Visualization. In *VIS '94: Proceedings of the conference on Visualization '94* (1994), IEEE Computer Society Press, pp. 117–123. 25

[PZ10] PALACIOS J., ZHANG E.: Interactive Visualization of Rotational Symmetry Fields on Surfaces. *IEEE Transactions on Visualization and Computer Graphics* (2010). 17, 43, 47

[Rot00] ROTH M.: *Automatic extraction of vortex core lines and other line-type features for scientific visualization*. PhD thesis, Technische Wissenschaften ETH Zurich, Zurich, 2000. 74, 91, 147

[RPH*09] ROSANWO O., PETZ C., HOTZ I., PROHASKA S., HEGE H.-C.: Dual Streamline Seeding. In *Proceedings of IEEE Pacific Visualization Symposium '09* (2009). 26

[SBH*01] SCHEUERMANN G., BOBACH T., HAGEN H., MAHROUS K., HAMANN B., JOY K. I., KOLLMANN W.: A Tetrahedral-Based Stream Surface Algorithm. In *Proceedings IEEE Visualization '01* (Oct. 2001), pp. 151–157. 17, 22, 25, 47

[SJ01] SATHERLEY R., JONES M. W.: Vector-city vector distance transform. *Computer Vision and Image Understanding 82*, 3 (2001), 238–254. 56, 57

[SJM96] SHEN H.-W., JOHNSON C. R., MA K.-L.: Visualizing Vector Fields Using Line Integral Convolution and Dye Advection. In *1996 Volume Visualization Symposium* (Oct. 1996), IEEE, pp. 63–70. 42

[SK98] SHEN H.-W., KAO D.: A New Line Integral Convolution Algorithm for Visualizing Time-Varying Flow Fields. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (Apr. – June 1998), 98–108. xiii, 8, 17, 42

[SLCZ09] SPENCER B., LARAMEE R. S., CHEN G., ZHANG E.: Evenly-Spaced Streamlines for Surfaces: An Image-Based Approach. *Computer Graphics Forum 28*, 6 (2009), 1618–1631. 17, 37, 40, 50

[SML03] SCHROEDER W. J., MARTIN K. M., LORENSEN W. E.: *The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics*, 3rd ed. Kitware, Inc., 2003. 144

[SP07] SADLO F., PEIKERT R.: Efficient Visualization of Lagrangian Coherent Structures by Filtered AMR Ridge Extraction. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1456–1463. 32, 33

[SRWS10] SCHNEIDER D., REICH W., WIEBEL A., SCHEUERMANN G.: Topology Aware Stream Surfaces. *Eurographics/IEEE Symposium on Visualization 29*, 3 (June 9–11 2010), 1153–1161. xv, 17, 29, 47, 62

[Sta] STANFORD UNIVERSITY: Aerospace Design Lab. http://adl.stanford.edu/docs/display/SUSQUARED/Tutorial+6+-+Turbulent+RAE+2822. Accessed: August 2013. xiii, 7

[STWE07] SCHAFHITZEL T., TEJADA E., WEISKOPF D., ERTL T.: Point-Based Stream Surfaces and Path Surfaces. In *GI '07: Proceedings of Graphics Interface 2007* (2007), ACM, pp. 289–296. xiv, 17, 21, 26, 46, 47

[SVL91] SCHROEDER W., VOLPE C. R., LORENSEN W. E.: The Stream Polygon: A Technique for 3D Vector Field Visualization. In *Proceedings IEEE Visualization '91* (1991), pp. 126–132. 24

[Swaa] SWANSEA UNIVERSITY: Marine Energy Research Group. http://www.swansea.ac.uk/engineering/marine-energy/. Accessed: August 2013. 11

[Swab] SWANSEA UNIVERSITY: Swansea University College of Engineering. http://www.swansea.ac.uk/engineering/research/. Accessed: August 2013. 100

[SWS09] SCHNEIDER D., WIEBEL A., SCHEUERMANN G.: Smooth Stream Surfaces of Fourth Order Precision. *Computer Graphics Forum 28*, 3 (2009). 17, 22, 27

[Tan] TANNOY: Tannoy. http://www.tannoy.com/. Accessed: 29th September 2010. 4

[Tec] TECPLOT: Tecplot 360. http://www.tecplot.com/. Accessed: March 2013. 5, 100

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'03* (2003), pp. 47–54. 88

[Tos] TOSCHI F.: International cfd database. http://cfd.cineca.it/. Accessed: 18th March 2014. xx, 96

[TS03] THEISEL H., SEIDEL H.-P.: Feature Flow Fields. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym 03)* (2003), pp. 141–148. 35

[TSW*05] THEISEL H., SHANER J., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Extraction of Parallel Vector Surfaces in 3D Time-Dependent Fields and Application to Vortex Core Line Tracking. In *Proceedings IEEE Visualization 2005* (2005), pp. 631–638. xv, 17, 35, 36, 46

[TvW99] TELEA A., VAN WIJK J. J.: Simplified Representation of Vector Fields. In *Proceedings IEEE Visualization '99* (1999), pp. 35–42. 66, 72, 88

[TWHS03] THEISEL H., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Saddle Connectors–An Approach to Visualizing the Topological Skeleton of Complex 3D Vector Fields. In *Proceedings IEEE Visualization '03* (2003), pp. 225–232. xiii, 9, 17, 18, 35, 46

[USM96] UENG S. K., SIKORSKI C., MA K. L.: Efficient Streamline, Streamribbon, and Streamtube Constructions on Unstructured Grids. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (June 1996), 100–110. 17, 24, 47

[VBvP04] VILANOVA A., BERENSCHOT G., VAN PUL C.: DTI visualization with stream-surfaces and evenly-spaced volume seeding. In *Joint Eurographics - IEEE TCVG Symposium on Visualization* (2004), Deussen O., Hansen C., Keim D., Saupe D., (Eds.), Eurographics Association, pp. 173–182. 50

[vFWTS08a] VON FUNCK W., WEINKAUF T., THEISEL H., SEIDEL H.-P.: Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2008) 14*, 6 (November - December 2008), 1396–1403. xvi, xx, 60, 96

[vFWTS08b] VON FUNCK W., WEINKAUF T., THEISEL H., SEIDEL H.-P.: Smoke Surfaces: An Interactive Flow Visualization Technique Inspired by Real-World Flow Experiments. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization) 14*, 6 (2008), 1396–1403. 17, 30, 31

[vW91] VAN WIJK J. J.: Spot noise-Texture Synthesis for Data Visualization. In *Computer Graphics (Proceedings of ACM SIGGRAPH 91)* (1991), Sederberg T. W., (Ed.), vol. 25, pp. 309–318. 17, 41

[vW93] VAN WIJK J. J.: Implicit Stream Surfaces. In *Proceedings of the Visualization '93 Conference* (Oct. 1993), IEEE Computer Society, pp. 245–252. 17, 18, 22, 34, 46

[vW02] VAN WIJK J. J.: Image Based Flow Visualization. *ACM Transactions on Graphics 21*, 3 (2002), 745–754. 43

[vW03] VAN WIJK J. J.: Image Based Flow Visualization for Curved Surfaces. In *Proceedings IEEE Visualization '03* (2003), IEEE Computer Society, pp. 123–130. 17, 43, 44

[War04] WARE C.: *Information Visualization: Perception for Design, Second Edition*. Morgan Kaufmann, 2004. 3

[WE04] WEISKOPF D., ERTL T.: A Hybrid Physical/Device-Space Approach for Spatio-Temporally Coherent Interactive Texture Advection on Curved Surfaces. In *Proceedings of Graphics Interface* (2004), pp. 263–270. 17, 26, 36, 44

[Wei07] WEISKOPF D.: Iterative twofold line integral convolution for texture-based vector field visualization. In *In Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration, Möller T., Hamann* (2007), Citeseer. 17, 42

[Wes] WEST, M.: Refactoring Game Entities with Components. http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy. Accessed: July 2013. 121

[WH06] WEISKOPF D., HAUSER H.: Cycle shading for the assessment and visualization of shape in one and two codimensions. In *Proceedings of Graphics Interface 2006* (Toronto, Ont., Canada, Canada, 2006), GI '06, Canadian Information Processing Society, pp. 219–226. xiii, 4, 17, 39

[WHN*03] WEINKAUF T., HEGE H., NOACK B., SCHLEGEL M., DILLMANN A.: Coherent Structures in a Transitional Flow around a Backward-Facing Step. *Physics of Fluids 15*, 9 (September 2003), S3. Winning Entry from the Gallery of Fluid Motion 2003. 50

[WJE00] WESTERMANN R., JOHNSON C., ERTL T.: A Level-Set Method for Flow Visualization. In *VIS '00: Proceedings of the conference on Visualization '00* (2000), IEEE Computer Society Press, pp. 147–154. xv, 17, 18, 22, 33, 34, 46

[WSE05] WEISKOPF D., SCHAFHITZEL T., ERTL T.: Real-Time Advection and Volumetric Illumination for the Visualization of 3D Unsteady Flow. In *Data Visualization, Proceedings of the 7th Joint EUROGRAPHICS–IEEE VGTG Symposium on Visualization (EuroVis 2005)* (May 2005), pp. 13–20. 81, 98

[WT02] WEINKAUF T., THEISEL H.: Curvature Measures of 3D Vector Fields and their Application. In *Journal of WSCG* (2002), V.Skala, (Ed.), vol. 10, pp. 507–514. 50

[WTHS04] WEINKAUF T., THEISEL H., HEGE H. C., SEIDEL H.-P.: Boundary Switch Connectors for Topological Visualization of Complex 3D Vector Fields. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym 04)* (2004), pp. 183–192. xiii, xv, 9, 17, 35, 46, 54

[XW05] XU R., WUNSCH D.: Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks 16*, 3 (2005), 645–678. 64, 88

[YMM10] YAN S., MAX N., MA K. L.: Polygonal Surface Advection applied to Strange Attractors. *Pacific Graphics 2010 29*, 7 (2010). 17, 28

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3D: An Interactive System for Point-Based Surface Editing. *ACM Transactions on Graphics 21*, 3 (July 2002), 322–329. 26

[ZSH96] ZÖCKLER M., STALLING D., HEGE H.: Interactive Visualization of 3D-Vector Fields Using Illuminated Streamlines. In *Proceedings IEEE Visualization '96* (Oct. 1996), pp. 107–113. 49, 50

APPENDIX A

Mathematical Concepts for Vector Fields

In this appendix we discuss a range of mathematical concepts used in the context of this thesis which are not previously described. We will start with some vector field basics, which will be followed by some cell interpolation schemes. Next we will describe how a spatial hash grid is used to speed up domain sampling. We follow this with spline interpolation schemes, and then the Runge-Kutta integration scheme. After this we describe some derived fields and finish this appendix with a table of notation.

A scalar field $s(\mathbf{p})$ is where every discrete point $\mathbf{p}$, within the spatial domain Ω, has a single real value $\mathbb{R}$. A vector field $\mathbf{v}(\mathbf{p})$ is where every discrete point $\mathbf{p}$, within the spatial domain, has a 3 tuple of real values $\mathbb{R}^3$ representing a vector quantity $\mathbf{v} = [\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z]$. Velocity data is represented as a vector quantity describing both direction and magnitude. For example, assuming a Cartesian coordinate system, a three dimensional steady state velocity field is described as $\mathbf{v}(\mathbf{p}) \in \mathbb{R}^3$ where $\mathbf{p} \in \Omega$, $\Omega \subset \mathbb{R}^3$, and Ω may be discretised as a structured regular grid/mesh, or an unstructured irregular grid/mesh.

Cell Interpolation Schemes Regular and unstructured meshes are a tessellation of the spatial domain using simple geometric structures. The mesh structures in this thesis are based on tetrahedrons or hexahedrons. These structures are referred to as cells, see Figure A.1. Each cell is an ordered combination of connected nodes. The ordering is impotent for correct interpolation and memory layout. Different CFD systems my use alternative orderings. Each node refers to a discretised point in the domain.

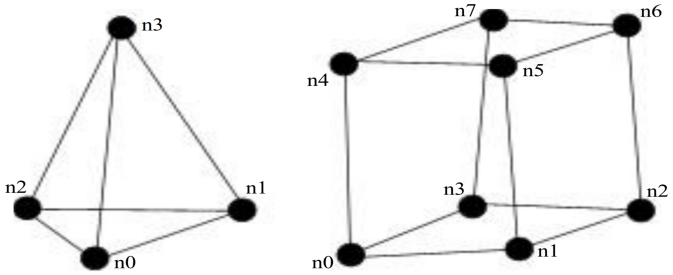


Figure A.1: *This illustration shows two cell types; a tetrahedron, and a hexahedron. The cells are constructed from ordered nodes $n(0...i)$. Each node is a discrete point in within the spatial domain.*

Interpolation of cells is required to deduce the value of attributes between each of the nodes. Referring to Figure A.1 the following equation is used for interpolation:

$$d = \sum_{i=0}^{n-1} W_i \cdot n_i \tag{A.1}$$

where d is the data to be interpolated and $W \in [0,1]$ is the weighting factor for each node, and where $\sum W_i = 1$. For a tetrahedron W is calculated as follows:

$$\begin{aligned} W_0 &= 1 - r - s - t \\ W_1 &= r \\ W_2 &= s \\ W_3 &= t \end{aligned} \tag{A.2}$$

and for a hexahedron W is calculated thus:

$$\begin{aligned} W_0 &= (1-r)(1-s)(1-t) \\ W_1 &= r(1-s)(1-t) \\ W_2 &= (1-r)s(1-t) \\ W_3 &= rs(1-t) \\ W_4 &= (1-r)(1-s)t \\ W_5 &= r(1-s)t \\ W_6 &= (1-r)st \\ W_7 &= rst \end{aligned} \tag{A.3}$$

where r, s, and t are the canonical coordinates of the cell, normalised to [0,1] [SML03].

Spacial Hash Grid The determination of the correct cell for interpolation is trivial for regular grids. A simple conversion from global coordinates to grid/parametric coordinates is required to identify the correct cell. Determining the correct cell to interpolate

for an unstructured grid however is not trivial. A brute force approach would be to search every cell in the domain and test if the point is situated inside. This approach is incredibly expensive. An alternative approach, utilised in this thesis, is the use of a spatial hash grid [EL07].

In summary we partition the domain into a hierarchy of virtual cubic grid cells of differing resolutions. These are associated to the irregular cells they bound using a hash function, and then stored in a hash table for fast retrieval. The virtual Euclidean grid consist of a potentially infinite number of cubic axis aligned grid cells. Each grid cell's edge length is defined as $k \in \mathbb{R}$. We map a certain grid cell resolution with a hierarchy using the subdivision level $l \in \mathbb{Z}$. l becomes a unique identifier for the hierarchical level of a virtual grid cell resolution. For every vertex within the domain we can derive a virtual grid cell resolution level l with the mapping:

$$\begin{bmatrix} \lfloor x/k \rfloor \\ \lfloor y/k \rfloor \\ \lfloor z/k \rfloor \\ l \end{bmatrix} \mapsto \begin{bmatrix} a \\ b \\ c \\ l \end{bmatrix} \tag{A.4}$$

For any tetrahedron or hexahedron c within the domain, the hierarchical level l is chosen such that the edge length k of the virtual grid cell optimally fits the size of c. k is optimal when c resides inside no more than eight virtual grid cells at level l. If $s = size(c)$ is the longest edge of the axis aligned bounding box of c we can define the hierarchical level l as follows:

$$l = \lceil log_2(s) \rceil \tag{A.5}$$

we can then define k as:

$$k = 2^l \tag{A.6}$$

The virtual grid cell size k is now defined such that c occupies virtual grid cells at the lowest possible level while never occupying more than eight virtual grid cells. We build the hash table of irregular cells using the following hash function storing a list of levels:

Algorithm XOR hash function.

```
1: function XORHASH(a,b,c,l)
2:     hash ↤ a x 73856093
3:     hash ↤ hash ⊕ b x 19349663
4:     hash ↤ hash ⊕ c x 83492791
5:     hash ↤ hash ⊕ l x 67867979
6:     return hash mod m
7: end function
```

Retrieval of the correct cell from the hash table is achieved by recalculating the hash of the vertex of interest, iterating through the list of stored levels until we find the correct cell.

Spline Interpolation Schemes There are cases when interpolation needs to be a smooth transition between vertices, which linear interpolation will not achieve. Cubic interpolation is one such method of producing a C^1 continuous curve. This thesis uses Hermite interpolation, specifically the special case of a cardinal spline known as a Catmull-Rom spline. This method calculates the tangents from the available vertices with lower computational cost, and thus ideal for our requirements. The computation of a C^1 continuous curve requires two vertices $\mathbf{p}_i$ which we interpolate between, and two tangents $\mathbf{t}_i$, one at each vertex. The Hermite scheme of interpolation requires the use of four basis functions h_j. The method of computation to find any point $\mathbf{p}$ along the interpolated curve at the interval $l \in [0,1]$ follows:

$$p(l) = h_0\, l\, \mathbf{p}_0 + h_1\, l\, \mathbf{t}_0 + h_2\, l\, \mathbf{p}_1 + h_3\, l\, \mathbf{t}_1 \tag{A.7}$$

The basis functions:

$$\begin{aligned} h_0 &= 2l^3 - 3l^2 + 1 \\ h_1 &= l^3 - 2l^2 + l \\ h_2 &= -2l^3 + 3l^2 \\ h_3 &= l^3 - l^2 \end{aligned} \tag{A.8}$$

The tangents for the Catmull-Rom spline:

$$\mathbf{t}_i = \frac{\mathbf{p}_{i+1} - \mathbf{p}_{i-1}}{2} \tag{A.9}$$

Runge-Kutta Integration A fundamental technique for the generation of integral lines or surfaces in a discretised vector field is the Runge-Kutta integration scheme. This method reconstructs a curve from a discrete set of tangents. The method used in this thesis is a fourth order double stepping adaptive integration scheme which provides a dense output in areas of high curvature [Hos94]. The refinement is achieved by specifying an error threshold beyond which the scheme will refine linearly. The idea is to determine the vector quantities $\mathbf{v}_i$ from an initial vertex location $\mathbf{p}_n$ and then compute a new vertex $\mathbf{p}_{n+1}$ in the direction indicated by the vector field. The step size or distance is defined by h noting that for the case of a velocity field h is the distance in time. For a steady state vector field, h is valid as we assume the vector field does not change over time, it remains steady. To compute the two half steps $\mathbf{p}_{n+1/2}$ and $\mathbf{p}_{n+1}$ we use the following formulae:

$$\begin{aligned} \mathbf{v}_0 &= \mathbf{v}(\mathbf{p}_n) \\ \mathbf{v}_1 &= \mathbf{v}(\mathbf{p}_n + \frac{h\mathbf{v}_0}{4}) \\ \mathbf{v}_2 &= \mathbf{v}(\mathbf{p}_n + \frac{h\mathbf{v}_1}{4}) \\ \mathbf{v}_3 &= \mathbf{v}(\mathbf{p}_n + \frac{h\mathbf{v}_2}{2}) \end{aligned} \tag{A.10}$$

$$\mathbf{p}_{n+1/2} = \mathbf{p}_n + \frac{h}{12}(\mathbf{v}_0 + 2\mathbf{v}_1 + 2\mathbf{v}_2 + \mathbf{v}_3) \tag{A.11}$$

$$\begin{aligned} \mathbf{v}_4 &= \mathbf{v}(\mathbf{p}_n) \\ \mathbf{v}_5 &= \mathbf{v}(\mathbf{p}_n + \frac{h\mathbf{v}_4}{4}) \\ \mathbf{v}_6 &= \mathbf{v}(\mathbf{p}_n + \frac{h\mathbf{v}_5}{4}) \\ \mathbf{v}_7 &= \mathbf{v}(\mathbf{p}_n + \frac{h\mathbf{v}_6}{2}) \end{aligned} \tag{A.12}$$

$$\mathbf{p}_{n+1} = \mathbf{p}_{n+1/2} + \frac{h}{12}(\mathbf{v}_4 + 2\mathbf{v}_5 + 2\mathbf{v}_6 + \mathbf{v}_7) \tag{A.13}$$

And to deduce the estimated error as a result of this step size we use this formula:

$$est_n = \frac{h}{72}(-\mathbf{v}_0 + 2\mathbf{v}_1 - \mathbf{v}_2 - 2\mathbf{v}_3 + 3\mathbf{v}_4 - \mathbf{v}_6) \tag{A.14}$$

Because the error est_n is linear we can simply divide est_n by the error threshold ε_{max} to find the new subdivision between $\mathbf{p}_n$ and $\mathbf{p}_{n+1}$ which will satisfy ε_{max}.

Derived Curvature Field From the steady state vector field we derive the curvature field [Rot00]. For any position on a curve or surface there is circle or sphere of radius r which approximates the local curvature. If the local curvature is straight then the curvature k is zero, and increases as r reduces from infinity. The curvature at any given location is defined as the reciprocal of the radius:

$$k = \frac{1}{r} \tag{A.15}$$

The curvature field $\mathbf{c}(\mathbf{p}) \in \mathbb{R}^3$ is derived by applying a combination of operators to the vector field. Steady state curvature is defined as:

$$\mathbf{c} = \frac{\mathbf{v} \times \mathbf{a}}{|\mathbf{v}|^3} \tag{A.16}$$

The curvature vector direction is the axis about which the vector field curves. The actual curvature k is the length of the curvature vector $|\mathbf{c}|$:

$$k = |\mathbf{c}| \tag{A.17}$$

The curvature field is derived from the first and second derivatives of the flow field. Where the velocity vector $\mathbf{v}$ is the first derivative:

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \end{bmatrix} \tag{A.18}$$

and the second derivative is acceleration **a**:

$$\mathbf{a} = (\nabla \mathbf{v})\mathbf{v} \tag{A.19}$$

$\nabla \mathbf{v}$ is the **Jacobian** of the velocity field, and is defined as:

$$\nabla \mathbf{v} \begin{bmatrix} \frac{\partial \mathbf{v}_x}{\partial x} & \frac{\partial \mathbf{v}_x}{\partial y} & \frac{\partial \mathbf{v}_x}{\partial z} \\ \frac{\partial \mathbf{v}_y}{\partial x} & \frac{\partial \mathbf{v}_y}{\partial y} & \frac{\partial \mathbf{v}_y}{\partial z} \\ \frac{\partial \mathbf{v}_z}{\partial x} & \frac{\partial \mathbf{v}_z}{\partial y} & \frac{\partial \mathbf{v}_z}{\partial z} \end{bmatrix} \tag{A.20}$$

where the gradient operator implies a vector of partial derivatives:

$$\nabla \Rightarrow \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \end{bmatrix} \tag{A.21}$$

Table of Notation Following is a list of mathematical notation used in this thesis. Each chapter is listed separately:

Chapter 1	
$\mathbf{v}$	Velocity field
ρ	Fluid density
∇	Gradient operator
p	Fluid Pressure
$\mathbf{g}$	Gravity vector
∇^2	Laplacian operator
ν	Kinematic viscosity
μ	Dynamic viscosity
ijk	Parametric coordinate system
xyz	Global coordinate system

Chapter 2	
$\mathbf{p}$	Steady state vertex
$\mathbf{v}_s$	Steady state velocity field
Ω	Steady state domain
$\mathbf{p}_t$	Time dependant vertex
$\mathbf{v}_t$	Time dependant velocity field
Ω_t	Time dependant domain
I_s	Streamline
s	Position along a seeding curve
t	Time

Continued on next page

Chapter 2 cont'd	
C	Seeding curve
S	Stream surface
I_p	Pathline
P	Path surface
T	Time interval set
L	Streakline
K	Streak surface
α	Transparency
$\alpha_{density}$	Density component of transparency
α_{shape}	Shape component of transparency
$\alpha_{curvature}$	Curvature component of transparency
α_{fade}	Fade component of transparency
$\mathbf{C}_{t_0}^{t}$	Cauchy Green deformation tensor field
$\mathbf{FTLE}_{t_0}^{t}$	Finite Time Lyapunov Exponent
λ_{max}	Maximum eigenvalue

Chapter 3	
$\mathbf{p}$	Steady state vertex
$\mathbf{v}$	Steady state velocity field
I	Streamline
t	Time
Ω	Spacial domain
Ω'	Spacial domain boundary
l_{max}	Maximum surface length
d_{test}	Minimum distance to nearest surface
d_{sep}	New surface offset distance
$\mathbf{v}'$	The velocity projected onto Ω'
$\mathbf{p}_{focus}$	Vertex of interest
$\mathbf{v}_{focus}$	Vector of interest
$\mathbf{v}_{stored}$	Stored vector
Ω_d	Distance field
$\mathbf{v}_{dist}$	Distance vector
$\nabla \times \mathbf{v}$	Curl of the velocity field (Vorticity)

Chapter 4	
$\mathbf{p}$	Steady state vertex
$\mathbf{v}$	Steady state velocity field
n	Cluster
Ω	Spacial domain

Continued on next page

Chapter 4 cont'd

ε	Error: total
a, b, c	Parameters of elliptic contour
ε_τ	Error: deviation from a velocity vector
$f(x_\tau, y_\tau, \varepsilon_\tau)$	Function of elliptic contour
$C_\alpha, C_\beta, C_\gamma$	Coefficients of elliptic contour
ε_ψ	Error: location
η_ψ	User controlled location coefficient
d, e	Aspect ratio parameters
$\eta_{\tau\psi}$	User controlled elliptic coefficient
ε_δ	Error: direction
ε_μ	Error: magnitude
η_μ	User controlled magnitude coefficient
η_δ	User controlled direction coefficient
s_l	Simplification level
l	Unique level
n_{root}	Root cluster
$\mathbf{c}$	Curvature field
Ω_c	Curvature field domain
$\mathbf{a}$	Acceleration field
∇	Gradient operator
$\nabla\mathbf{v}$	Jacobian of the velocity field
vol	Spacial volume
$length$	Spacial length
s	Seeding curve
α_c	Transparency: curvature dependant
α_v	Transparency: view dependant
α	Transparency: total
$\hat{\mathbf{n}}_\mathbf{v}$	View normal
$\hat{\mathbf{n}}_\mathbf{s}$	Surface normal
η_α	User controlled transparency coefficient
RGB	Red Green Blue final colour vector
RGB_{in}	Red Green Blue input colour vector
η_{RGB}	User controlled saturation coefficient

Chapter 5

Ω	Spacial domain
$\mathbf{p}$	Steady state vertex
$\mathbf{v}$	Steady state velocity field
$\mathbf{c}$	Steady state curvature field
$\mathbf{g}$	Steady state gradient field

Continued on next page

Chapter 5 cont'd	
$\mathbf{a}$	Steady state acceleration field
∇	Gradient operator
$\nabla\mathbf{v}$	Jacobian of the velocity field
k	Number of clusters
s	Scalar value
c	Cluster centroid
π	Cluster partition
$\mathbf{e}$	Euclidean distance vector
i	Linear relationship of curvature and velocity gradient
l	Linear relationship of i and distance
B	User controlled curvature or velocity bias coefficient
A	User controlled B or distance bias coefficient
d	Distance function
vol	Spacial volume
$length$	Spacial length

APPENDIX B

Gallery of User Options

In this appendix we demonstrate the algorithm defined in Chapter 5 applied to the Bernard dataset with a range of A and B parameters. The motivation for this gallery is to demonstrate the variation of the results with change in parameter values. We assume the ideal parameter for k is 8 due to pre knowledge about the number of features contained within this flow field. We first show the seeding curves generated from the clusters to more easily identify the change in location as a result of parameter changes. These differences quite subtle in some cases.

The seeding curves are coloured red, and there length is a function of cluster volume as discussed in Chapter 5. The arrow glyphs represent the average vector of all vectors within the cluster subset. The arrow glyph base represents the centre of the cluster, which is also the centre of the seeding curve.

Following the seeding curves we show the generated surfaces colour mapped to vorticity in the range $[0, 1.66]$, where 0 is mapped to cyan, and 1.66 is mapped to yellow. The surfaces are rendered with transparency mapped to curvature, with view dependant saturation switched on. Streamlines are also rendered to the surface, coloured red to contrast with the colour map.

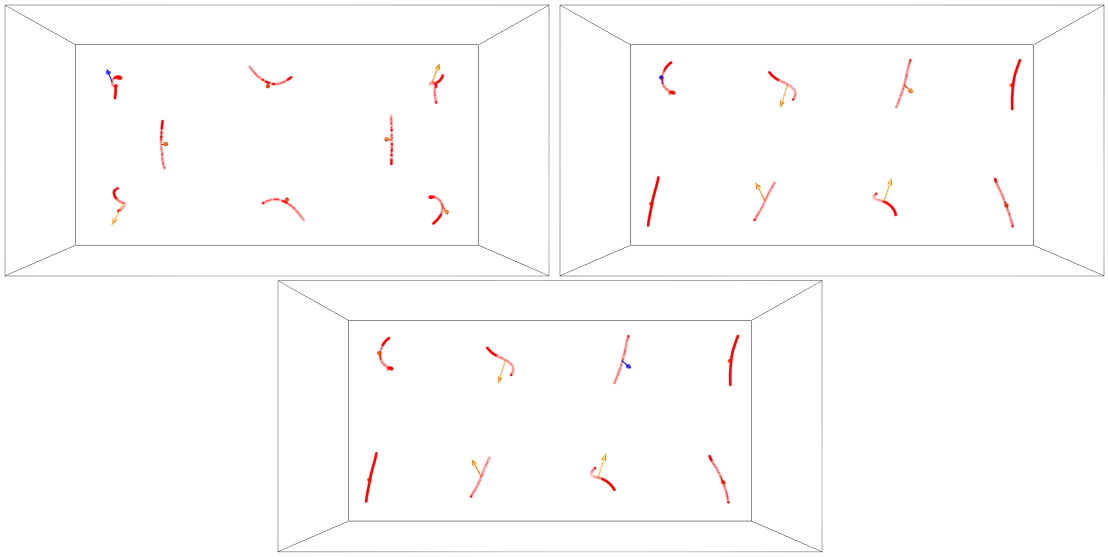


Figure B.1: *Top left parameters are k=8 and A=0.9 and B=0.1. Top right parameters are k=8 and A=0.5 and B=0.1. Bottom parameters are k=8 and A=0.1 and B=0.1. The seeding curves are coloured red. The arrow glyphs represent the average vector of the cluster; its base is located at the centre of the cluster.*

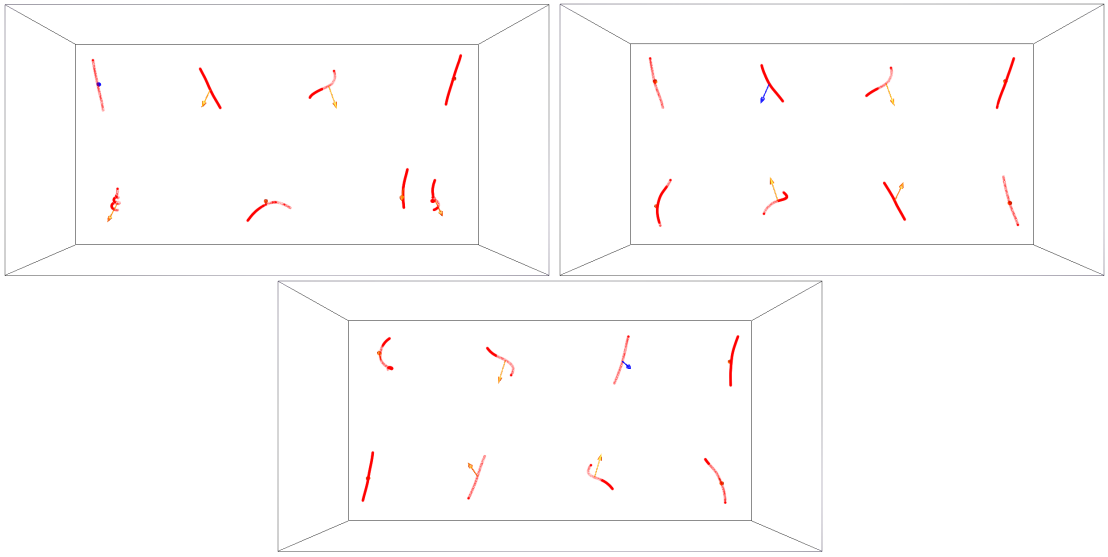


Figure B.2: *Top left parameters are k=8 and A=0.9 and B=0.5. Top right parameters are k=8 and A=0.5 and B=0.5. Bottom parameters are k=8 and A=0.1 and B=0.5. The seeding curves are coloured red. The arrow glyphs represent the average vector of the cluster; its base is located at the centre of the cluster.*

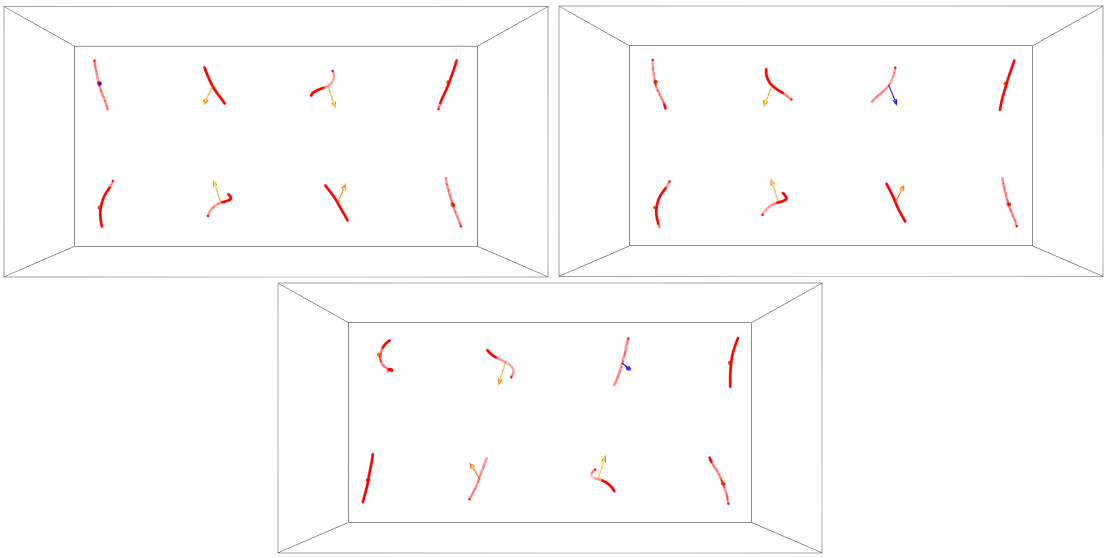


Figure B.3: *Top left parameters are k=8 and A=0.9 and B=0.9. Top right parameters are k=8 and A=0.5 and B=0.9. Bottom parameters are k=8 and A=0.1 and B=0.9. The seeding curves are coloured red. The arrow glyphs represent the average vector of the cluster; its base is located at the centre of the cluster.*

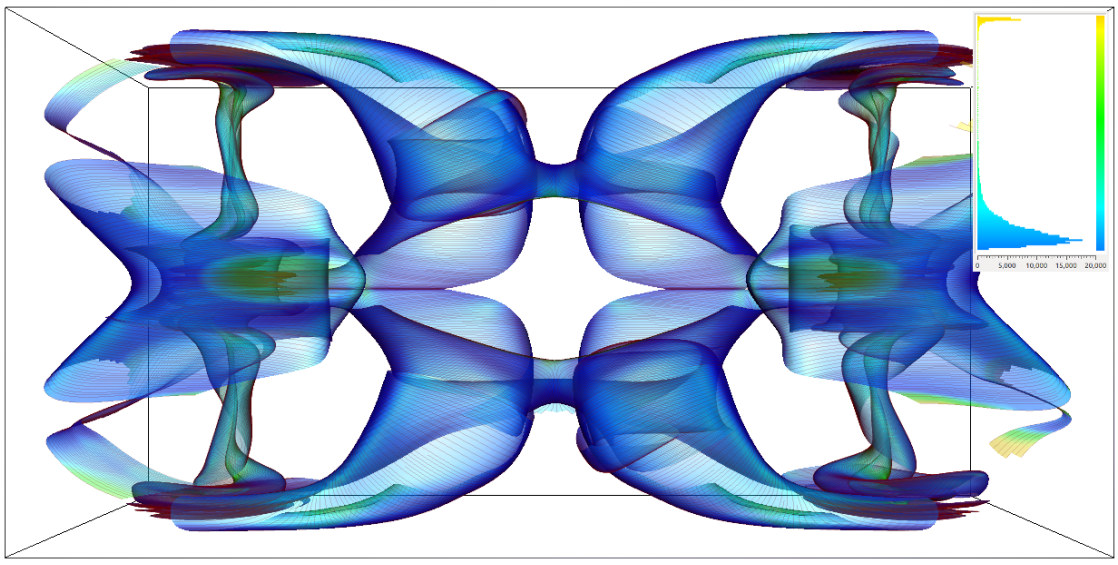


Figure B.4: *This visualisation parameters are k=8 and A=0.9 and B=0.1. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

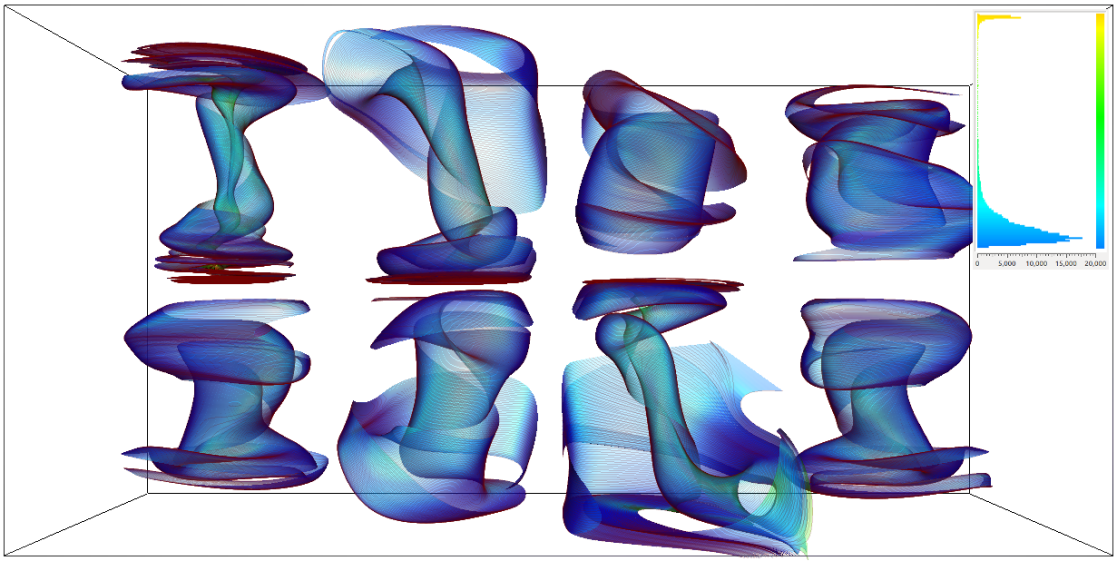


Figure B.5: *This visualisation parameters are k=8 and A=0.5 and B=0.1. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

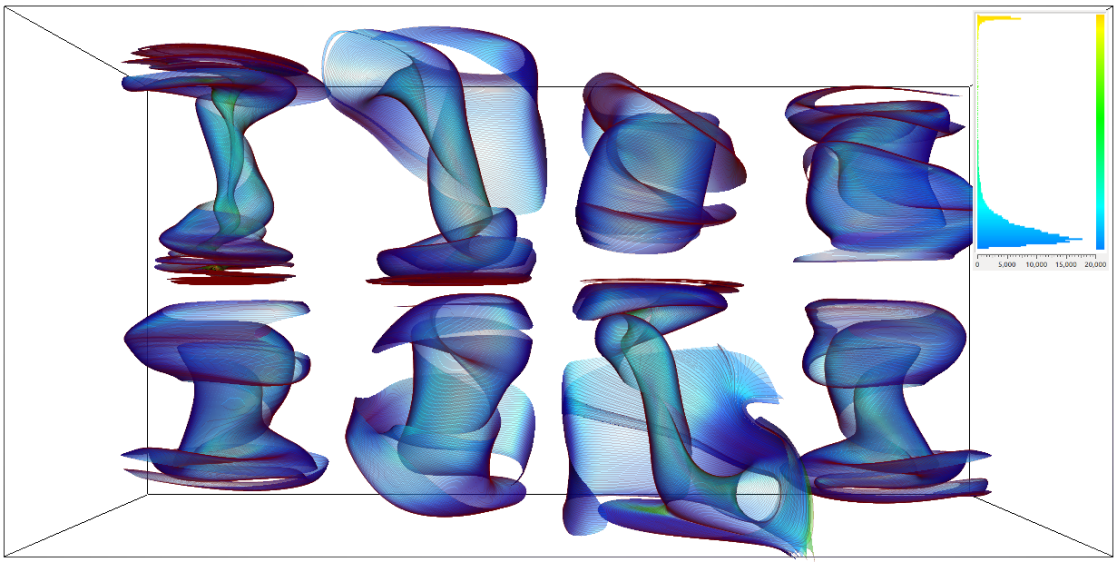


Figure B.6: *This visualisation parameters are k=8 and A=0.1 and B=0.1. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

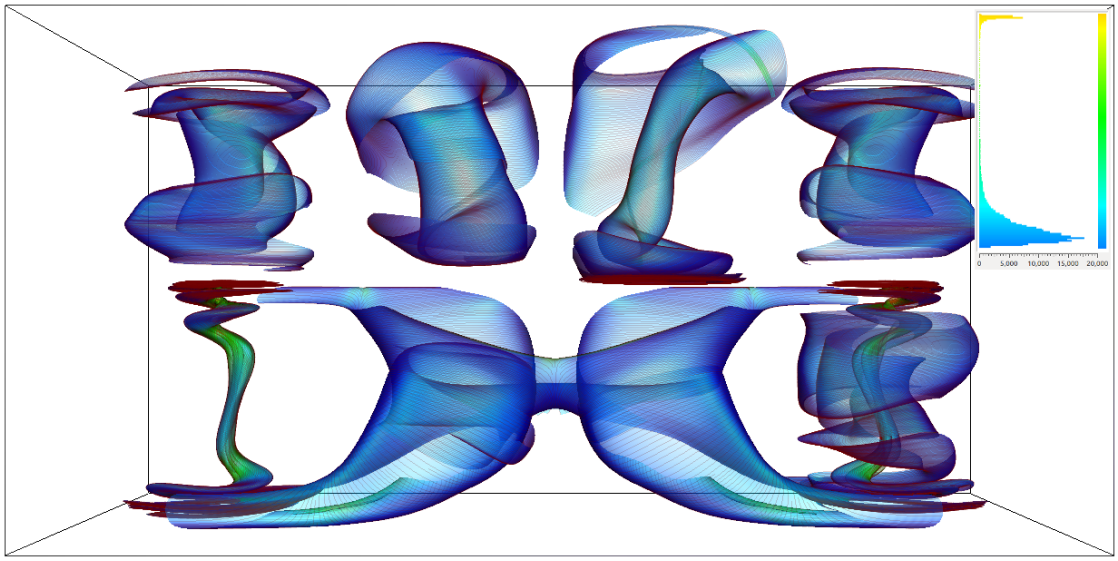


Figure B.7: *This visualisation parameters are k=8 and A=0.9 and B=0.5. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

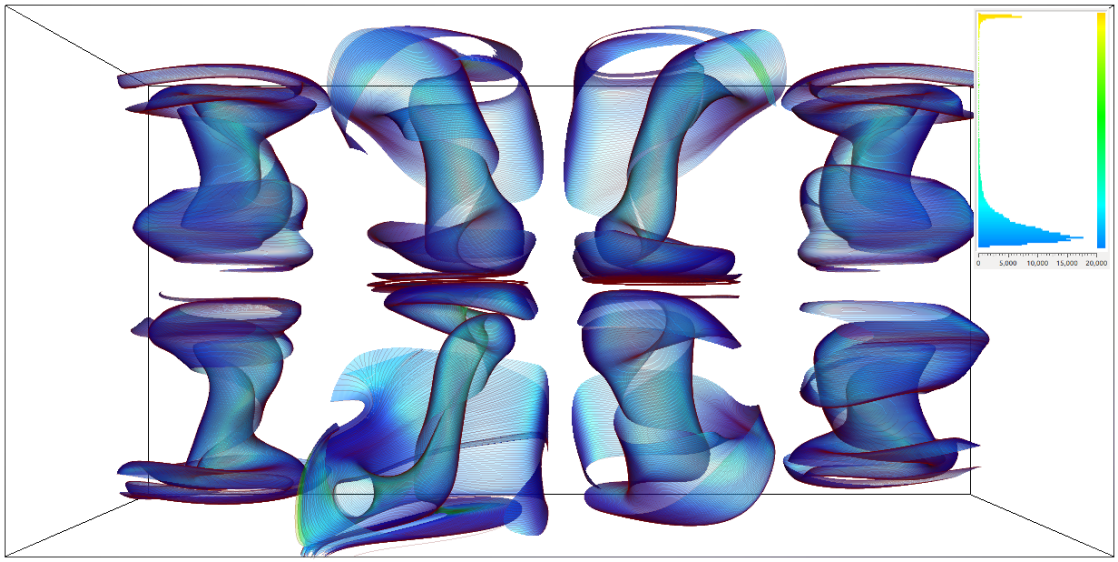


Figure B.8: *This visualisation parameters are k=8 and A=0.5 and B=0.5. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

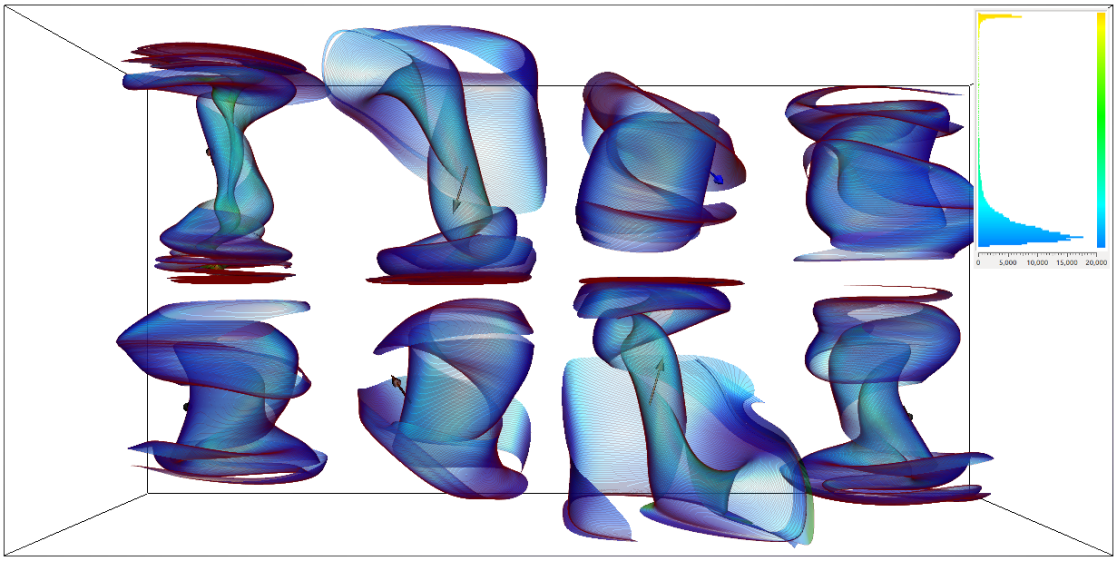


Figure B.9: *This visualisation parameters are k=8 and A=0.1 and B=0.5. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

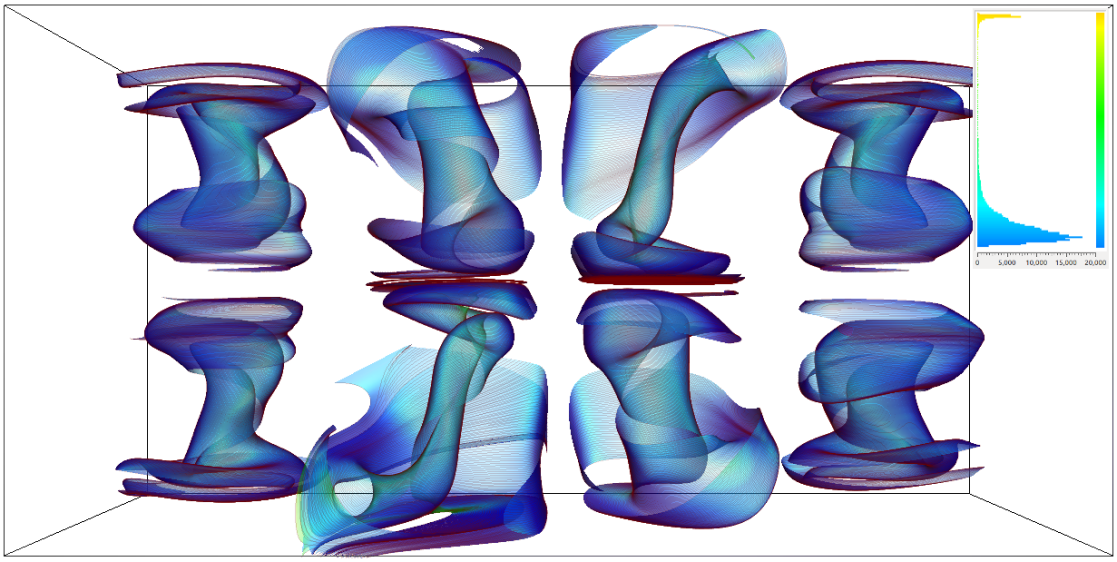


Figure B.10: *This visualisation parameters are k=8 and A=0.9 and B=0.9. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

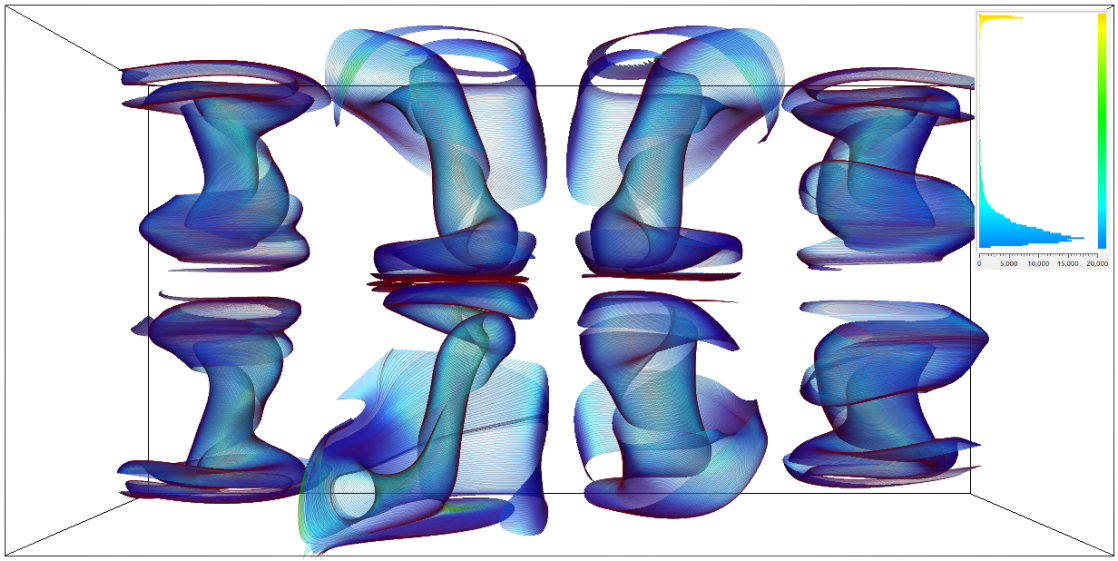


Figure B.11: *This visualisation parameters are k=8 and A=0.5 and B=0.9. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*

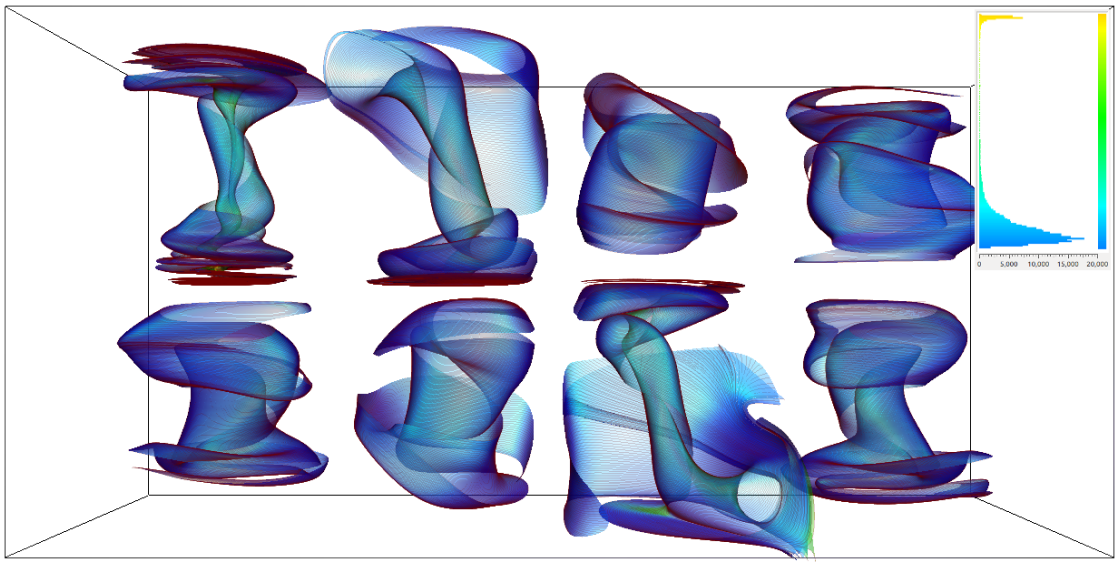


Figure B.12: *This visualisation parameters are k=8 and A=0.1 and B=0.9. The surfaces are rendered with transparency, view dependant saturation, and streamlines.*