Stian Sivertsen

Ridge Extraction, and Illustrative Visualization of an FTLE Flow Field.

Master Thesis

supervised by

Hellwig Hauser

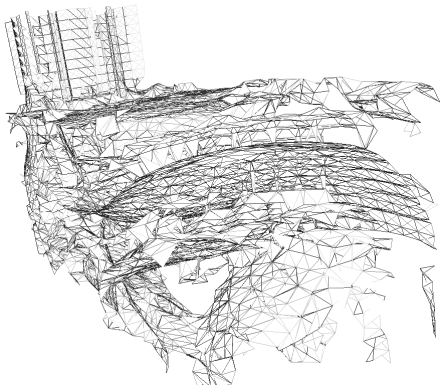
Andrea Brambilla

Institute of Informatics

University of Bergen

Abstract

Illustrative visualization of a flow is an area that is proving to be interesting and in later years something that have been somewhat explored, it can provide some very interesting results and in this thesis we find the required structures needed for such a visualization. Lagrangian coherent structures are now commonly used in flow visualization and we establish a simplistic approach to creating them in this thesis. The problem of occlusion and the general complexity of both the data and the visual results are still a common thing when working with a structured flow. This thesis presents a solution that identifies the complex and often large amounts of data that is inherent in a flow and tries to find a good way of extracting the structures within. By using the FTLE values found in the flow data we define these structures so that we can see what is going on in these complex systems, as well as combining it with an illustrative approach that will provide an interesting visual aspect to the integral structures found in flows. By using mathematical analysis of the raw data we obtain the necessary tools we need to define and extract ridges that form LCS that we can then model. We use a simple approach by focusing on the basic components of such complex structures as well as utilizing often long standing principles in an interesting way. Combining the strengths of both illustrative visualization and the clarity of integral structures we can create a more visual sound model that loses some of its cluttering complexities but is still capable of generating a pleasing visual end result.



Contents

CHAPTER

1 Introduction

Begin at the beginning and go on till you come to the end: then stop.

Lewis Carroll,

1.1 Finding A Solution

There are many applications in science where visualizing intricate parts of a model can be difficult. Complex structures and delicate flows often makes it hard to extract the components that are taking place within the structures. In the later years newer technologies and methods have appeared that help leviate this problem, by separating and extracting structures using a selection of algorithms that focuses on finding correct and interesting information within complex structures. In this thesis we will use different techniques and structures to aquire a reasonable result from a given dataset that will ultimately help us in defining the occurences and changes that takes place in complex flows.

An important structure within the context of this field are lagrangian coherent structures, or LCS for short. These structures help us define and separate the different regions in the flow that our datasets are based on. The underlying component of an LCS are finite-time Lyapunov exponents, or FTLE. These exponents

help define the separation between the elements of the flow and by extracting certain components based on the FTLE in the flow we can define LCS that give us a good idea not only of the different regions of the flow, but also what is happening inside these regions.

In this thesis we will utilize datasets that have precalculated FTLE values at every point recorded in the flow, and use this value to evaluate and extract ridges that define the overall LCS in the flow. This structure is eventually processed and visualized so we can better understand the underlying components of the flow.

Visualizing the end product of all the calculations and structures is equally important because this defines the product that describes what is actually happening in the regions. There are many different methods for visualizing the result we get, and they all focus on different aspects of the structure. As we will show, extracting a coherent structure from the dataset is only a step in figuring out what exactly is going on in the flow. Although there are many visual approaches that could have been made in finalizing the results, we chose to focus on illustrative visualization as there are many different areas within this field that helps extract the smalller components and intricate structures within bigger complex datasets - as well as producing some very nice visual end results.

Having big datasets and different components poses many difficult questions as to how to approach the problem. Deciding upon a software to visualize, or a method to extract the required data are all very important questions to answer. In this thesis we decided to approach the data using already established methods as well as using a custom built framework based on C++ and openGL to produce the end results. Using commercial software for debugging and verification purposes only.

In this thesis we aim to use existing structures as well as methods to visualize highly complex flow data based on the FTLE component in the flow. After defining the structures using ridges, we create a triangle mesh as a final structure that we will use custom built shaders based on existing work to visualize the different aspects of the model so that we can produce interesting and intricate results that give a new perspective on the raw data.

1.2 Thesis Outline

The second chapter is divided into different subsections each defining and explaining the ground principles used within this thesis. Here we try to not only explain the use and necessity for the many different structures and algorithms we are basing our work on, but also referencing other work that either contributes directly to the foundation of our solution or that have an interesting or perhaps similar approach to getting the results we desire. It does however keep the explanations brief and is mostly a high level overview of the different aspects used in this thesis, so it is recommended to browse through the referenced material for further information on the different areas. Chapter three presents the theory of our solution as well as defines some of the boundaries set for what we are trying to achieve. We also discuss how we chose our solution as well as some interesting aspects of the different components of that solution. The fourth chapter delves into the actual implementation of the proposed solution, and describes the technical choices made to create the end results, as well as explains what is happening in more detail.

Results are then presented in the fifth chapter, giving a visual representation of what we achieved as well as defining some of the end produce of the implementation and how well it performed on the different datasets. And how far we came towards our goal. At the end of the thesis we look into some of the conclusions and what we gained from the solution. The impact or degree of results as well as looking at some interesting alternate paths or future work.

CHAPTER

2 State of the Art

All cases are unique and very similar to others.

T.S. Eliot

2.1 FTLE And Lagrangian Coherent Structures

Finite Time Lypanov Exponent

The Finite Time Lypanov Exponent is the basis of all of the calculations and values in the thesis. This is the value in the data that defines the outcome of our solution. The FTLE is applicable to time dependant discrete data sets. The flow systems that are discretely defined as such often have a chaotic nature to them that might be hard to visualize internally and it is then very helpful to have an exponential value that describes the state of the discrete flow as to base a computer generated visualization on so that we can better grasp the intricate flows and structures that are produced within the datasets.
FTLE is an often used and valuable measurement for analyzing the behavior in unsteady flows. Although it can be hard to prove its accuracy or to measure the rate of success and likeness to structures in the flow because there is no ”ground truth” available to measure. It has however proven to be quite valuable and accurate in its representation. There have also been work done to comparatively measure its

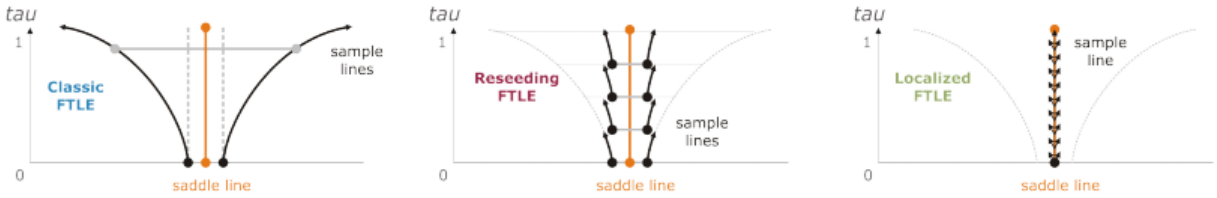


Figure 2.1: Illustrating 3 different methods for computing the FTLE. [22]

success and performance by Kuh et al.. [22]. In their definition of what the FTLE does they note that it ”measures the rate of convergence or divergence between neighboring flow particles over a finite time interval.” Of the resulting scalar field that is produced from computing the FTLE value over a vector field, ridges are one of its most useful and significant structures. These structures closely resembles and fits with the definition of lagrangian coherent structures.It is often then that the FTLE is used to define ridges that in place construct lagrangian coherent structures, or LCS. These structures define separate regions of the dataset that have a similar enough component as to coexist as a whole unit. There are a variety of ways to compute the FTLE over a vector field, each process having both negative and positive components bound to computational complexity and resulting accuracy. Three of the most common methods of finding the FTLE can be seen in figure 2.1 where we see an illustration of the classic method, the reseeding approach as well as the localized approach that rely on jacobian values.

It can be seen as a relative measure but it has still become a standard way of observing the separating behavior within unsteady flows. There are many different areas that is has been used on. Examples includes describing flow behavior on a planetary surface [4] [11], the movement of jelly fish [18],[49] and even in turbine separation analysis, just to name a few.

The computation of FTLE is based on the spatial gradient of a calculated flow map. The main aspect that different methods for finding the FTLE differ is in the acquisition of this flow map.

Kuhn et al. gives a benchmark for evaluating FTLE computations in this paper[22]. Here he discusses the uses for the FTLE as well as different meth-

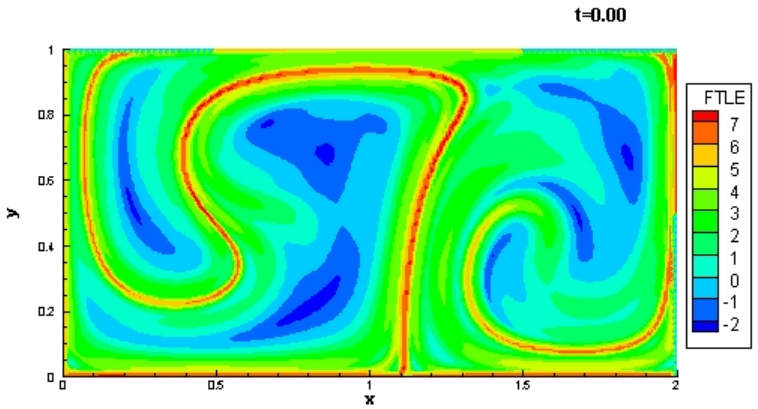


Figure 2.2: The double gyre : an often used example of an FTLE defined flow [41]

ods for obtaining it, each having distinct performance and accuracy trade-offs. It also goes into more detail on how FTLE has helped us understand the complex flow behavior in unsteady flows. The paper also describes how to better compare and verify the end result of a FTLE computation.

Lagrangian Coherent Structures

LCS are structures created by segregated components, often from dynamically distinct regions of a flow. The structures help reveal geometry and phenomenas that are otherwise hidden when viewing just the vector field, or a single trajectory in the flow. It is a way to gain an overview of a very complex set of structures created by a chaotic flow often based on an advanced function. These separate regions in flow can be described by the previously mentioned FTLE.
There have been a number of attempts to find something that describes the structure of an unsteady flow field in the same way that topological skeletons describe a steady flow - using either lines or surfaces. The most prolific way of doing so has been using LCS defined by ridges identified in the FTLE scalar field. This provides a very effective way of visualizing both complex and simple flow fields. This has in turn caused the fluid dynamic community to embrace the LCS. However a drawback with the LCS is the computational cost of producing them. Although most algorithms are very easily parallelized, and they perform very will

in this manner, they still require quite a long time to produce results. One of the more popular approaches is to use a ridge tracking algorithm.

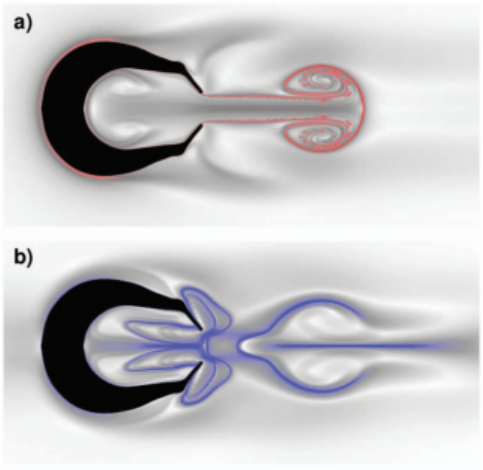


Figure 2.3: Showing backward(a) and forward (b) LCS for Sarsia tubulosa.

Haller and Yuan [13] introduce a lagrangian definition to represent the boundaries of coherent structures in two dimensional turbulent flows. These boundaries represent lines that are responsible for stretching the passive tracers, and they have been using this to study coherent vortexes. Shadden et al. [42] define LCS from FTLE values much the same as we do in this thesis, however they focus more on the representation of the LCS - where they want to show that the flux generated for a LCS is negligibly small. Green et al. [10] also uses the same concepts to create LCS from FTLE values to study fluid mechanics. Here they study three dimensional flows like a vortex and compare the results to Eulerian methods. They find that ”Despite additional computational costs, the DLE(FTLE) method has several advantages over Eulerian methods, including greater detail and the ability to define structure boundaries without relying on a preselected threshold”

Another very nice paper on extracting LCS from both steady and unsteady flows have been written by G. Haller et al.[12] where they find coherent structures in three-dimensional flows. Here they use FTLE values as a successful approach for getting the structures directly from particle paths in the flow.

2.2 Ridges And Ridge Extraction

Ridges and valleys define what constitutes as the main components of a surface. The mathematical definition of a ridge as a curve has been known for quite some time and ridges have been heavily used in geomorphology. As mentioned by Peikert and Sadlo "In Image analysis and computer vision, a digital image can be seen as a sampled scalar field or height field, making ridges available as characteristic structures complementary to the more popular edges." So ridges can be seen as a valid form of defining interesting regions by representing the dataset as a height field and then identifying the local maxima areas in this height field as a ridge. These ridges then combine to define a resulting structure.In essence a ridge is defined as a single curve and a 1-dimensional entity, but they can be extended to other dimensions, as done by Kindlman et al.[19] where they used ridge surfaces to visualize MRI data.

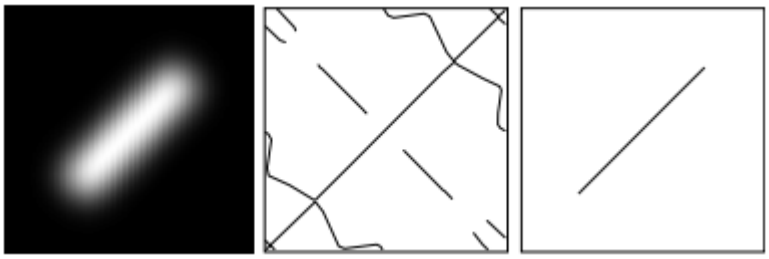


Figure 2.4: A representation of a single Ridge [25]

Assortment Of Ridges

There are many different variations of how a ridge can be defined as well as conflicting views as to what constitutes as a ridge. The most common ridge used is the simple height ridge. In the paper by Peikert and Sadlo [30] a height ridge is defined by its second derivative values. This equates to a height ridge as a set of points that define a local maxima where the direction of the ridge is the direction of the maximum second derivative q and the orthogonal axis of minimum second

derivatives p. Giving the definition of a height ridge according to Eberly [3]:

$$\begin{aligned} Lp &= 0 \\ Lpp &< 0 \\ Lpp &\leq Lqq \end{aligned} \tag{2.1}$$

This is the ridge type that we focus on in the solution but there are also other ridges that vary in complexity.

Second derivative ridges are found in a similar way to height ridges, but it uses higher derivatives to define it. This leads to a number of differences from the height ridges, most apparently the number of ridges that are found giving it a more refined response.

C ridges is also a variant and is a further evolution of the standard height ridge. The C-ridge differs from the height ridge in that it uses the major eigenvector as a defining component instead of defaulting to the minor eigenvector of the hessian, as is done with a height ridge. This ridge can also be extended to 3D by utilizing the local FTLE maximum. For more information about the use and applicability of C-Ridges Schindler et al. [38] is a recommended read. For more detailed information about the definition of the height ridge as well as the second derivative ridge the thesis by Majer [25] is a good read. There are however different approaches to define a ridge that are valid, so finding an exact definition is not always that simple. This mostly affect the number of ridges found and how coarse they are - and it also depends on the filters and directions used to detect the ridges.

Applicability Of Ridges

Ridges, represented as a set of curves are often used to define important geometric information within structures. They have been used in many different aspects of image analysis as well as computer visualization. This pertains to both medical as well as flow visualization. In the paper by J.Sahner Et Al [35] they utilize height ridges to help define topological separatrices of vortices and strain skeletons. Sadlo and Peikert [34] visualize components of flow inside turbines, and more generally LCS that are computed from grids of trajectories. Another example is Shadden and Lekien [42] where they use ridges to find LCS that define

flow separation. In medical visualization there have been work done using ridge surfaces, most notably by Kindlmann et all. [19].

Ridge Extraction

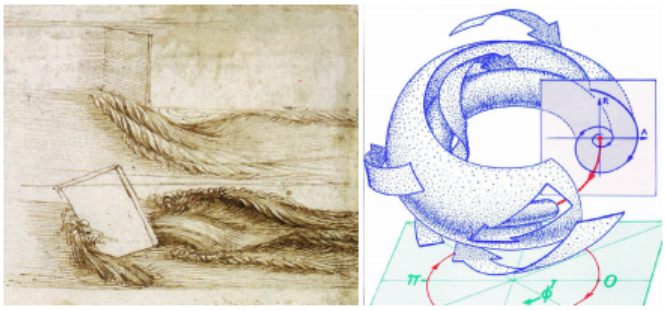
After the ridges have been separately defined it is often important to find the structures that they create as a whole. This is done after detecting the ridges by the FTLE values and validating that the ridges actually exist and represent the data correctly. In most cases when building LCS from ridges the approach is to create a seed point in the data and then iteratively expand the ridges in the appropriate direction to get a general structure that is defined by neighboring ridge points. There have been many algorithms and methods as to figure out how to best extract the ridges, and it is not always easy. Extraction depends a lot on the available data, as well as finding a suitable way to reduce the computational cost of finding neighboring points.

The method used for extracting ridges are often based on what information is easily available from the data points, as well as the wanted dimension and structure of the found ridges. A simple approach to extracting ridge lines is using the parallel vectors approach described in the paper by Peikert and Roth [29]. Here they utilize vector fields and an operator to get the resulting lines. One of the more general approaches to extracting ridge points from a dataset is using marching ridges [6]. This method is based on the marching cubes algorithm and it uses a set of criteria to define the different ridges that can be found depending on user defined parameters and dimensions of the data. It then uses a seeded approach where it iteratively expands a found ridge point by locating neighbors and segments the data into a valid neighborhood. Another approach for ridge extraction is the AMR Filtered approach used by Sadlo Peikert [34]. Here they subdivide the dataset into neighboring cells and then grow the ridges in the appropriate neighborhood based on the filters and criteria set for a valid ridge. They also use a look ahead approach, as to not neglect smaller components of the ridge structure.

Lipinski and Mohseni [24] uses a tracking algorithm to define the LCS that are created by the resulting ridges. Here they focus on finding a better computational approach to massive amounts of complex data by using approximation and

estimation using tracer particles.

2.3 Illustrative Flow Visualization



(a) (b)

Figure 2.5: Hand-drawn illustration of water flow behind an obstacle by Leonardo da Vinci. (b) Depiction of a dynamical system with stream arrows by Abraham and Shaw

The combination of the two techniques produce a large existent field within visualization as some of the previous examples have shown. Although using an illustrative approach to this kind of data is in no way new. It has been used to describe scientific phenomenons for as long as 500 years. In more recent years we have an attempt to use hand drawn pictures to describe flow structures found in Abraham and Shaws paper [1].

To better describe the mixing of two disciplines Brambilla et al STAR [2] has created a very nice categorization of the areas within both flow and illustrative visualization. The categories are defined are:

1. Raw data - original data produced by simulations or measurements.
2. Integral structures - linear structures used to describe the flow.
3. Flow features - the relevant data as it pertains to the user.

To better understand the different areas of both flow visualization and illustrative visualization they have been separated into two subcategories, but both of them focusing on flows and the structures within them.

Flow Visualization

Integral surfaces are surfaces that are created of separate regions. In the case of LCS the different ridge surfaces created by the ridge extraction is combined to define an integral surfaces that is required to create a proper visual representation of what is happening in the dataset. The ridge lines produced, be it 1 dimensional curves or 2 dimensional surfaces tell interesting and intricate tales of what is happening within the flow of a dataset.

Integral curves have been used in visualizing these complex concepts for a long time, and have proved invaluable to simplify everything that can happen within a chaotic flow. Integral surfaces take the concept of streamlines and streak lines to another level. Computing the surfaces of advanced structures can be very expensive but with the latest development in technology the added flexibility and visual results created by using integral surfaces have been more than worth it.

Integral structures first entered the research field when Hultquist [15] proposed a way to represent particles moving through a flow as a surface by using a stream surface algorithm. There are many different ways to represent an integral surface using both a triangle mesh as well as particles. Some even utilize point based rendering - and work has been done on both CPU and GPU to try and speed up the often very costly surface generation that is required for advanced flows. The popularity of integral structures can be attributed to the ability of these structures to create a clear representation of the trajectories of particles in the flow.

Integral lines was the first attempt to harness the usefulness of these structures done by Schroeder et al. [40] where an n-sided polygon is swept along a streamline and is deformed to local flow properties. Ueng et al. [45] extended this to work with unstructured grids and Schirski et al. [39] tries to speed up the process.

When increasing the dimension of the structures we get problems with self occluding and the like. Therefore as we move into the second dimension new approaches had to be carved out to better represent the data inherent in the flow. This resulted in a focus on visibility issues instead of more perceptual ones.

Expanding into further dimensions have not been a priority, mostly because the increased complexity is not justified by the result. There has however been some attempts at it and Xue [48] visualizes streamvolumes using a texture advec-

tion technique.

Post et al. [31] categorized the flow visualization techniques into four main groups, and in the last few years a fifth group has also been suggested by Salzbrunn et al. [36]:

1. Direct Visualization - maps the data directly to a visual representation. Low complexity.
2. Texture-based Visualization - Use local flow attributes to create a noise texture. Further information found in [37] and [23]
3. Geometric Visualization - Uses integral structures as a basis. More information can be found in the paper of McLoughkin et al. [27].
4. Feature-based Visualization - Focuses the visual result on the most important aspects of the vector field. Details found in Post et al. STAR[31]
5. Partition-based Visualization - Tries to effectively partition the spatial and temporal domain using flow properties.[36]

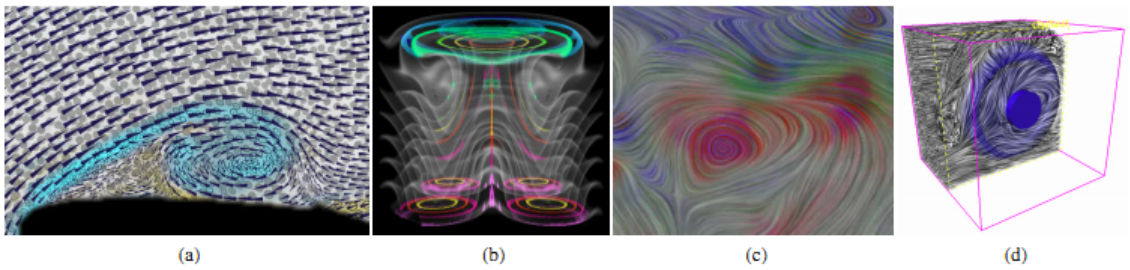


Figure 2.6: (a) The visualization method described in [20] uses concepts from painting to visualize 2d incompressible flows: arrows represent velocity, colors represent vorticity and ellipses represent strain, divergence and shear. (b) Illustrative volume rendering of flow by Svakhine et al. [44]. (c) Texture-based visualization with color-coding of local flow properties [46] (d) 3D-LIC of flow around a wheel, visualized with the aid of a clipping plane [32]. Image taken from STAR [2]

There has also been attempts at utilizing a focus+context approach to integral structures. There have been 3 major areas that have been explored in this regard.

And they all provide different but interesting goals. In Fuhrmann and Gröllers paper [5] we see a user-applied focus. Here they have a magic lens or a magic box that the user employs to enhance and focus the visualization so that what they perceive as important is kept in focus. Another approach based on Hauser and Mlejnek [14] has the focus aspect on the seeded regions within the flow. This approach is further explored in [26] and [21]

The third category focuses on entire integral curves in the visual result. Jones and Ma [17] did just that by presenting a flow exploration framework that allows the user to select the focus+context streamlines that are then enhanced and displayed. Wei [47] created an interface where the users first sketches the most interesting streamline shape and similar streamlines are identified within the data and then set in focus.

The field of flow visualization has been around for many years but there are still a lot of unanswered questions. Some of the challenges present are finding more effective ways of producing and generating the data as it can be quite time consuming. This makes it difficult to have any sort of interaction with the data as a lot of preprocessing is required. The complexity and organization of the data also gives different problems in regard to the grid the data is stored in. There is a gap between structured and less structured grids that provide a problem when trying to process and visualize data.

Also the amount of data present in a flow makes it hard to avoid and reduce clutter when the visualization goes into three dimensions. The data itself and the constructs it creates also makes it so that twists,overlapping folds etc. are present. And as more areas within the different categories are explored we also see that the complexity is ever increasing within this field so finding solutions that reduce the complexity but still keeps the visual result intact, or better yet improves it are venues that researchers are very interested in. Garth et al. [7] describes an approach for generating integral surfaces in time-dependent vector fields. The method described uses surface approximation and a graphical representation to directly compute the surfaces. Hummel et al. [16] also uses integral surfaces when implementing illustrative rendering to enhance and produce structures that describe what is happening in complex flow structures. They go on to describe integral surfaces as ”ideal tools to illustrate vector fields and fluid flow structures.”

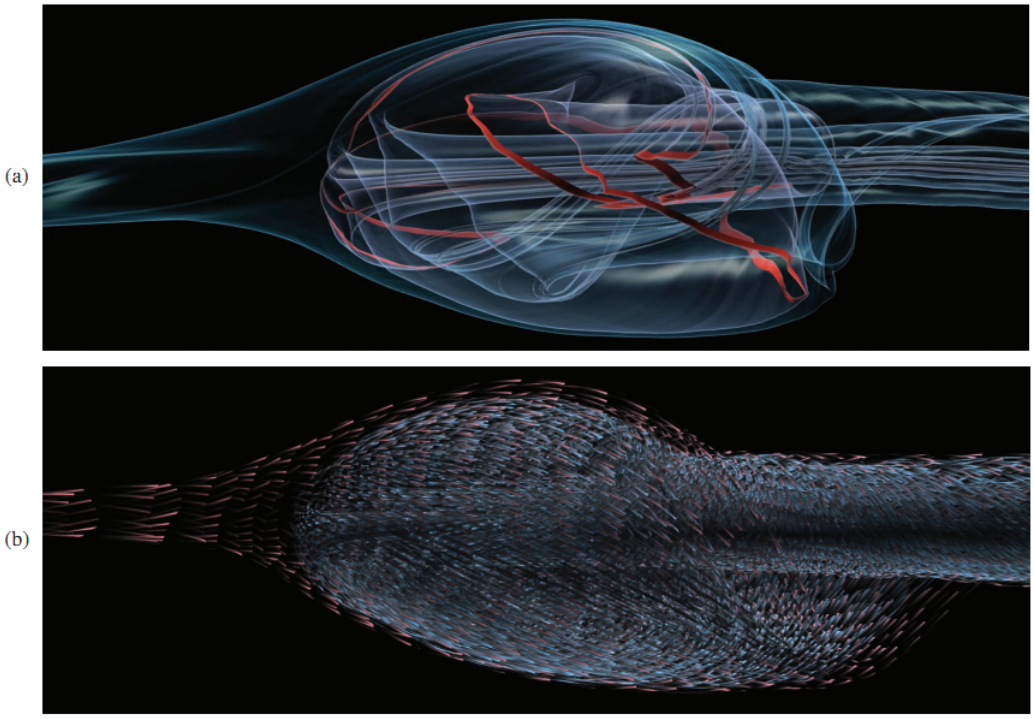


Figure 2.7: A stream surface visualizes flow inside a vortex breakdown bubble. In (a), the surface is rendered with strong normal variation transparency and light silhouettes. The opaque red stripe illustrates the front of the surface. In (b), a modulated stripe texture conveys the impression of dense particles traces; here, flow direction is indicated by intensity modulation, and velocity is expressed as the length of the traces. Images taken from [16].

Illustrative Visualization

There are many ways of representing a structure or visualize a model. Photorealistic visualization is often used, especially for models and components that have a real life counterpart as to create a bigger sense of realism. Illustrative visualization is based on a more artistic approach to representing data. Here the main concept is to use artistic techniques that have been refined and helpful throughout history to simplify and focus the context on the important aspects of the model. As mentioned previously we also have different aspects of illustrative visualization. The main focus is to use the illustrative techniques, often in the sense of shaders

and transparancy to simplify or refocus the structure so unecessary structures and obstructive components are removed, or minimized.

A very good example of this is again the paper by Hummel et al. [16]. Here they generate surfaces from a turbulent jet dataset and render them using different illustrative techniques as to enhance the understanding of what is happening within the flow represented by the surface. For more specific techniques used within illustrative visualization there is Gooch et al [8] explaining how to use tone shading to keep edge lines and highlights to give a clear picture of what is being represented. There is also a paper by Sousa et al. [43] that explains the method of graphite pencil rendering as well as a paper citeSousa2003 that accurately reveal the geometric forms that give subjects their characteristic shape.

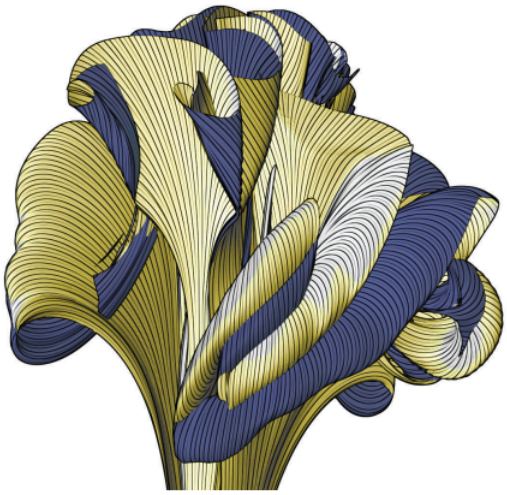


Figure 2.8: A path surface generated from a turbulent jet dataset, rendered using an adaptive stripe pattern. [16]

There are very few techniques that uses illustrative approaches to deal with flow features in a focus+context approach compared to the areas explored with integral surfaces. There has however been some work done by Muelder and Ma [28].

More related to surface rendering Gorla et al. [9] study the effect of textures as a way to represent the line orientation of the surface flow. This helps the viewer follow the flow in a more static manner, so as to not be overwhelmed by everything

that is happening - but still have access to all the information present. For a more general approach to volume illustration Rheingans and Ebert [33] describes not only the benefits of utilizing a nonphotorelaistic approach but also gives a collection of methods that extend common NPR techniques to volume objects. Using photorealistic rendering as a counterpart when explaining the process.

CHAPTER

3 Presenting The Solution

Once upon a midnight dreary.

Poe

3.1 What Are We After

The basis for this thesis is to take components from existing areas within computer visualization. Starting by examining the different qualities of advanced flows, and the difficulties inherit within this data a number of valid solutions proved apparent. Having big datasets comprising of time dependant flow can prove challenging to examine, and especially figuring out and internally visualizing all the components that are within such flows is a challenge.

Often the structures and components within these flows are so obscure and abstract that finding real world components to base a resulting visualization on, as well as having it provide sufficient detail and informative properties so that the end gain is substantial enough can prove difficult. There is also the question of how to approach the different data structures when composing a solution, as to best utilize the different aspects and data structures of the data we are handling.

In this thesis however the main goal of the solution was to use an existing technique on new and interesting data, and seeing if it was possible to produce similar positive results with different data, then what had been previously tested.

Defining and handling the data structures became an integral part of the solution as we moved forward with testing. It also proved apparent that it was important to gradually escalate the complexity and dimensions of the solution as we moved forward because of all the different components, and the sensitivity of not only the properties of the data but also the intricate parts that needed to be established before the next step could be made.

Going over this we focused on having a simple and well established data structure - basing the solution on LCS that were created from ridges extracted from a pre- generated flow dataset with existing FTLE values. Now the goal of the solution is to not to only execute a satisfying way of extracting the necessary components for a useful visualization of structures within a complex flow, but also finding a good visual technique for presenting the end results. The visual technique that proved most interesting for this purpose was illustrative visualization. As mentioned earlier there had already been some success with going non photo-realistic when composing complex data structures, specifically in the way details and important objects in the data was preserved.

This all resulted in the solution being a gradual 2d to 3d representation of a pregenerated flow dataset with designated FTLE values. These FTLE values then of course form a height field that is solved with the emphasis on finding valid LCS structures using a form of ridge extraction that would enable us to create a triangle mesh that we can utilize some form of illustrative visual technique on to produce satisfying results as to containing components of the flow, and to preserving the more intricate details of the dataset.

Having read and been inspired by a paper by Hummel and Garth [16] that not only provided valid results of advanced flows based on integral surfaces, also gave very interesting and almost artful results of complex structures that we wanted to execute and replicate on other structures - or flows.

3.2 Motivations

Now be able to generate a valid solution to the problem at hand we needed ways to:

1. Define and store the data to be worked on.
2. Validate and refine the data so it can be properly processed.
3. Retrieve the structural components required for a visual representation of the data.
4. Apply a visual technique or component to the resulting structures, as to enhance its internal structures and properties.
5. Compare the resulting structures with existing similar results - for verification.

Thinking about the concepts and motivation behind the solution we can see that there are many problems that arise when working on bigger sets of complex data. Delving into the problem without first looking at all the possible choices to make and what the implications of these choices are can be difficult. This resulted in a decision to use pregenerated datasets with all the necessary and validated information for producing the end results. A choice made also on the basis that the main asset of this thesis is not on the actual data generation and validation but more on the resulting visual experience.

Handling the data and verifying that it is correct as well as that it captures all the important qualities of the data we used is of course an important part, but having to also calculate and validate the data that the resulting structures are based on would delude the focus of the solution. Instead finding a valid and often used way of capturing the internal structures of the data was important.

This led us to utilizing ridges as it is a often used concept when generating some form of model of a flow. This also led us to utilizing the FTLE values within the flow as it proved very good information on the overall structure as well as provides a very fluid way of extracting good structures of lagrangian nature, using some form of ridge extraction that will result in the desired model we are after.

It was important to verify each step of the process, so not only was the filters and other methods based on well tested solutions, but it was also an important step to provide a first step into the process as to validate the initial data. This is

why we first started creating a 2 dimensional solution of the 3d flow. This enabled us to verify the different aspects of the data and our algorithms in a less complex environment before extending the resulting algorithm to three dimensions.

Seeing as most ridge extracting techniques are based around a seeding and neighbourhood expanding we had to come up with a different approach to generating the resulting structure of our flow since we already had all the necessary information stored within all points in our dataset and just wanted to extract or grasp the actual structural components that these data points defined.

So after defining the data we had to work on in a valid structure, and then validate and refine it using well known techniques like applying a sobel filter and using different mathematical analysis and algorithms to extract all the necessary information we needed to create and retrieve the structural components. First in a simplified 2d case and then later extend the solution into three dimensions we also needed a valid way to construct the end results into a workable model.

There are many different ways not only to represent the structures within a flow - but also many different approaches as to how we can obtain them. Initially we thought we could modify existing techniques used for ridge extraction and surface construction. So the goal was to take a process that used a seed and grow approach and adapt it to instead just extract the structure directly using some of the same principles. This proved difficult however seeing as the structure of our data made it then difficult to assign the found components to their respective regions as well as finding a valid way to create the entire structure. Instead we used the analytical part of one of the proposed solutions, the AMR Ridge Extraction by Sadlo and Peikert. [34] We then used the data found with the mathematical analysis to incorporate the a marching ridges approach [6] to the structure. This also proved difficult seeing as this also uses a more expanding approach to building ridge surfaces - however with a slight modification on how to define the structures found resulted in the desired result that we could then proceed to visualize using our illustrative approach.

Again being able to validate the information gathered along the way was an important aspect of our solution and we utilized third party software to validate our resulting model before starting the more visual process of our solution.

After we were happy with the resulting model we started to contemplate on some of the different visual approaches we had discussed on using. Keeping with the previous stages of our approach to this implementation we chose to focus on well defined and validated approaches that could be implemented in a simple way and that would provide a satisfiable and testable end product. This resulted in focusing on implementing a custom shader that would extensively enhance the inherent properties of our model.

3.3 Structure

So the end result of what we have is a valid solution that take pre-existing techniques and well established approaches to the problems faced and place them in an we then define them in such a way that we can retrieve the desired end result of our data. The first step in our solution becomes defining an import function based on the dataset, where we are able to store all the important pre-generated values of the data. This helps us organize the flow in a logical way and it is therefore important to make it easy to apply the filters and analysis on the data that we plan on doing. This results in a n-dimensional array that organizes the flow and its enclosures in a grid fashion so that every coordinate of the flow is easily accessible.

After we have organized the data we need to find a structure for applying the appropriate filters and analysis. Most well known filters for these kind of data are organized in a grid or box fashion, like for instance the sobel operator. This makes them ideal for our data since we have already organized it in a grid fashion running through the array in either a two-dimensional or three-dimensional procedure becomes trivial. This is also the case for more mathematical analytics, seeing as most is done by an approximation filter, and not by calculation to ease up on the complexity and cost of the algorithm. Storing the results of these analysis also becomes trivial seeing as we have access to each individual point in the data, and this makes it possible to also update and store new data at each given point.

Retrieving the structure from the then processed data points becomes a matter

of adapting an algorithm that correctly recognizes the building blocks we are after, in our case we are looking for ridge lines for 2d and ridge surfaces for 3d. This equates to curves in 2d and a triangle mesh in 3d. Now as mentioned most of the predefined algorithms use a seed and expand approach so we need some way to identify the construction of each cell and then create the appropriate structure - and then in the end combining them all together to form the resulting model that is represented by the combining surfaces.

After we have the model we need, we need to custom build our shaders and apply them directly to the model to produce our end visual result. Refining and adding post processing effects as needed. With this end product we can now check it for consistency and validity. Either by comparing it to existing results or by the validity of the original data that we know is correct.

The illustrative part then comes last as we have to define a way to better view our model in a way that it makes sense and enhances the structures within the flow. We therefore need to construct an illustrative approach that makes sense for the complex data and model we have obtained. Post processing the model and the data is also an option as we can further refine our results seeing as one of the bigger issues when it comes to flow visualization is general noise and hard to read data. There is also the problem that most flow visualizations face which is the fact that the complex data often occludes and makes it hard to see what is going on.

This is why we need to apply a shading technique that addresses these issues while still maintain the simplistic nature of our approach.

So our solution is then condensed into these separate steps to fulfill our initial assumption:

1. Store the data as data points in a logical array representing the entire dataset.
2. Apply filters and mathematical analysis on the data to extract the required information we need.
3. Use an existing modified algorithm to construct our model from the analyzed data.
4. Apply a custom built shader on the resulting model to give us our visual end result.

5. Verify our end result versus pre-existing models, and the initial data.

After we have obtained a valid and formed model and applied our illustrative approach to it we then have to expand our datasets and validate the method so that it works for all different kinds of flow structures based on the same principles that we have in our test data. We also need to verify that our approach works on real data, however this could be more related to future work as well as further exploration of these types of data. There are many different and interesting data that can be obtained and that can prove interesting for further validation.

CHAPTER

4 Implementation

If I have ever made any valuable discoveries, it has been owing more to patient attention, than to any other talent.

Sir Isaac Newton

4.1 Defining The Solution

As mentioned earlier we decided to make the solution from scratch using common libraries and frameworks. Primarily we used openGL for the graphical representation and Qt for the framework. Writing the solution in C++ and using ShaderMaker for initial testing of the initial Illustrative Visualization Part.

There were also several choices available both in representing the data and extracting the different components that we need to properly visualize the results and get the solution we wanted. The more common ways of creating LCS consisting of ridge components are by seeding and expanding areas of the data and then joining it together to form a coherent structure. Seeing as we had all the initial values precomputed we only wanted to structure the data logically as well as providing easy access and a coherent way to execute extraction of structures,

applying filters and processing the data to conform into the values and structures we needed for the end result.

Seeing as the data can become very complex and that many different aspects and parts of it are used as parameters in the algorithms and mathematical analysis to define the model we need it was needed to first create a 2D result that would limit the complexity and in the minimal dimension possible remove all the artifacts and define the proper algorithms and preprocessing needed and still keep the data and result coherent so that we can better perform error checks and validation of the data.

This is why we first elected to produce a simple height map of the 3d data in 2d slides to get an overview of the data and to help decide not only the correct approach to identifying the ridges in the data but also the best way to structure the data and model containing them as well as the best algorithms for constructing the structures defined by the ridges and how to best store the information found. When we were happy with the ending results in the minimal dimensions we expanded the implementation and its algorithms and analysis to a third dimension and then compare this result to the 2d to verify the correctness of the model.

To easier understand the implementation and information when describing the two main approaches taken when implementing the solution we will first go over some of the key components and aspects of the implementation and briefly describer their relevance and use in the final implementation. We will then refer to these concepts and go into greater detail within both the 2D and 3D implementation and how they were utilized to create a cohesive and sound end result.

We organize the data that forms the flow in a grid structure for easy access and navigation. Giving it the dimensions X,Y, and Z given by the restricting boundary of the flow structure compartment. This also allows us to easily switch from 2D to 3D as the 2D case is accessing slices in one direction.

In each data point we store the precomputed components we have acquired like flow velocity, FTLE value,position etc. recognized by either filters or analysis like the gradient and the transverse direction. We also decided to divide the sections of the data into something called grid elements. These grid elements represent an appropriate dimensional structure for the overall dimensional result. This means that for a 2D result we would have a slice comprising of grid elements

represented as squares of 4 data points. Within this square there can exist ridge points that in the end define a ridge line.

These ridge lines are then bound together automatically in the entire slice when they are singularly identified within each grid element. Now this creates the main goal of our implementation, use the precomputed data structured in our grid and define grid points that eventually gives us the ridge lines we need to define a proper Lagranghian Coherent Structure. To do this we employ mathematical filters and analysis, principally using convolution to apply filters to the data in its appropriate dimensional forms that results in good approximations for the first and second derivatives, or the gradient and hessian values needed to identify ridge points within the data.

These points are then stored as interesting and tested by analysing the eigen properties to existing principals that define different ridges. As we focus on height ridges we need the notable data points found to adhere to the principals found in the definition of a height ridge. This means that it has to be a local maxima in the height field. When all the ridge points have been acquired within our dataset we then use an appropriate method, depending on the dimension of our result, to construct the ridge constructs that give us our final model.

These ridge constructs are then represented in a triangular mesh that define our model consisting of LCS. Finding this construct is pretty straightforward in the lower dimensions as we only connect ridge points using lines. It gets a bit more complicated when expanding it to 3D as we have to create appropriate triangular compositions within each grid element that makes sense and does not contort the resulting model. We use the marching ridges algorithm as a starting point for this.

To summarize the general approach for the implementation after we have organized all the data from the flow is:

1. Convolute the data using an appropriate filter to get the derivatives.
2. Perform an eigen analysis to get values required for detecting ridges.
3. Go through the data and detect ridge points using the values gathered and finally create ridge lines.
4. Extract the model from the ridge lines that have been found.

4.2 The 2D Solution

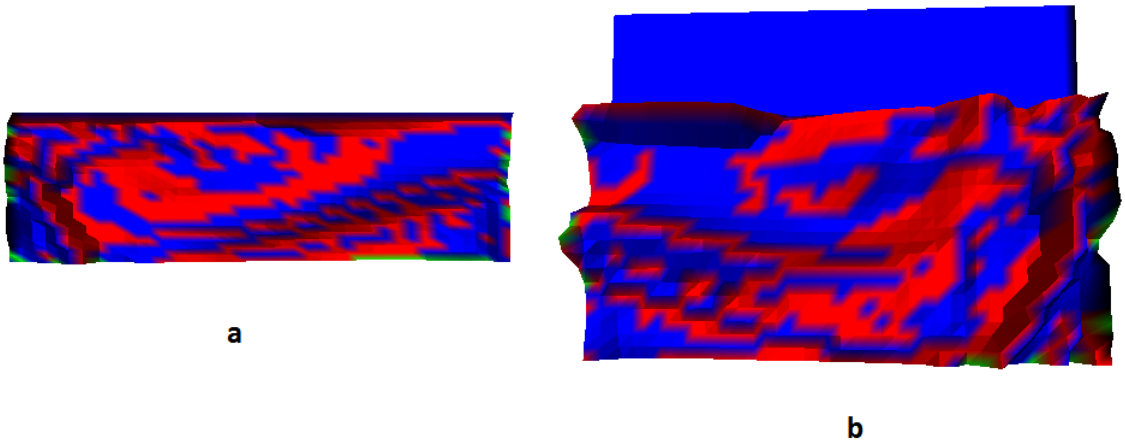


Figure 4.1: A simple visualization of a 2d slice Using a color code for the FTLE values from low blue to high red. a)Early instance b)Later stages

We started at the lower dimensionality so that we can better understand the process and make it easier to not only internally visualize the correct result but also easier and faster to improve the method and reduce errors. The grid elements used in the 2D case are squares defined by 4 connected edges. Each edge has a possibility of identifying a ridge point in what is known as a zero crossing.

Now as we started implementing the solution we wanted to focus on clarity when approaching the 2D. This way we could have a good overview of how the flow behaved and how the FTLE values formed the slices in the data. As a first approach we then displayed the slices of the flow as a height map defined by it's FTLE values. This can be seen in figure 4.1. Here we can see an indication of the ridges in the slice as they form peaks in the slice. This served as reference images for later algorithms and data processing as we could always compare the results on a visual basis with the raw data.

After a simple visualization we needed to process the data so that we had the necessary computations needed for detecting ridge points and ridge lines. Ridge points are defined as possible areas in each grid element (In the 2D case we have 4 possible locations for a ridge point) that can contain a valid ridge point as defined

by the height ridge. So in essence these ridge points, once detected and verified would serve as the component when finding the ridges we need to get our resulting model.

To compute the correct ridge points we looked into a number of approaches utilized earlier but we landed on an algorithm that takes the convoluted data as well as the values found in eigen analysis and uses the core principals found in the marching ridges algorithm to recognize a possible ridge point and verifying it by comparing the gradient,hessian and transverse direction to predefined values that gives us a valid height ridge.

Starting with the convolution we apply a sobel filter that gives us the gradient and hessian at every point in the data. The gradient is estimated using the filter and gives us an indication of the change at every point in the height field defined by the FTLE values. The Hessian gives us an indication of the curvature. These values are used to further identify a height ridge. Before we can go through the data and identify height ridges we still need some more information. This is done in a second step where we identify the eigenvalues and vectors associated as well as finding the transverse direction. The transverse direction is used when extracting the ridges. This is to make sure all the ridges are ordered properly when combining them into a final structure.

Going back to our first step, there are many variations and filters that can be used to compute the derivative values needed. As we are utilizing height ridges we found that one of the more simple and well known approaches would be to use a sobel operator to estimate these values. Computing them individually would not be cost effective. And in our 2D case we create matrices of each slice that hold the actual values at each index comparable to where the data point is stored. In this way we can quickly access the relevant data for each part of the model in a slice by slice basis.

For our 2D case we then get our gradient $G$and our hessian Has our first step, going through a single 2D slice in the data and for each valid non-empty data calculating G and H.

$$G = \begin{bmatrix} Dx \\ Dy \end{bmatrix} H = \begin{bmatrix} DxDx & DxDy \\ DyDx & DyDy \end{bmatrix}$$

These are all identified using the sobel operators defined below. These values are then stored in each data point for quick access when needed in later calculations.

Sobel filters used for 2D:

$$Dx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} Dy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$DxDx = \begin{bmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{bmatrix} DxDy/DyDx = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} DyDy = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix}$$

The second step in our implementation is then to go through the slice again and this time identify the eigenvector corresponding to the smallest eigenvalue. This eigenvector is then used to check the transverse direction of that point. The eigenvector is again stored in the data points and we can now begin identifying ridges seeing as we have all the necessary information needed.

The definition of a ridge, in our case a height ridge is given by a couple of criteria. The information we have gathered so far can go along way to identify potential ridges within the data but we need a set criteria that defines, in our case, the height ridge so that we can fully verify it as a ridge and make it part of our final model. Seeing as the algorithm we use is based on marching ridges it is executed in a way that it identifies a ridge defined within a grid element by verifying the second derivatives of a zero crossing on the edges of that grid element. In the 2D case a grid element consists of 4 edges and an edge consists of two endpoints in our dataset. This means that an edge can be a member of more than one grid element so they are therefore marked in the implementation so to not be checked more than once for a valid ridge point. So for each edge we check the transverse direction against the average transverse and we do this by finding the greatest eigenvector from the 2x2 matrix C. This matrix is defined as $C = \frac{1}{2} \times \vec{v} \times \vec{v}^T$. We then check our current transverse direction vs the average and adjust if the

dot product of our stored eigenvector and the newly found eigenvector of C is negative.

After this is done we have to identify if there exists a zero-crossing on the edge, and if so where it is. This is done by interpolating between the first derivatives or gradients at each endpoint of the edge. So there can only exist a zero-crossing if the first derivatives of the two sides have opposite signs and when this is the case we have a possible ridge point on our edge.

To then check that the zero-crossing we found is in actuality a ridge point we have to check the second derivatives at this location, which is again interpolated using both the transverse direction and the hessian we can interpolate to find the second derivatives at the possible ridge point. If this interpolated value is less than zero we have identified a correct ridge point and it is added in a collection that in our 2D case will eventually form a set of curves that represents our ridges as a set of lines. We also define the magnitude of the second derivative at each ridge point so that we can describe the convexity at each ridge point. This convexity is used when forming ridge lines within grid elements when there are more than one choice for connecting the ridge lines formed by several points in a grid element.

After we have defined and validated all the ridge points in a slice we then have to go through each grid element and define the lines that connect them. In our 2D case we need a way to connect the two ridge points in a way that it makes sense for the resulting construct. For exactly two ridge points the ridge line is trivial, and a line is defined between them. For the cases where 3 and 4 ridge points are contained within a single grid element we need some way of connecting the ridge points that makes sense.

It is here that we utilize the convexity that we calculated when validating ridge points. In this case we connect the two ridge points that are most convex. Here the convexity s between point a and point b is defined as $s = aC \times bC$.

These lines are then added to define the ridges found in a 2D slice and can be represented visually.

The method described here in the 2D case was done to establish the founding steps needed to create construction of height ridges that defined the FTLEs in a flow. Some example images are given below to illustrate the results and displaying both ridge points and ridge lines found.

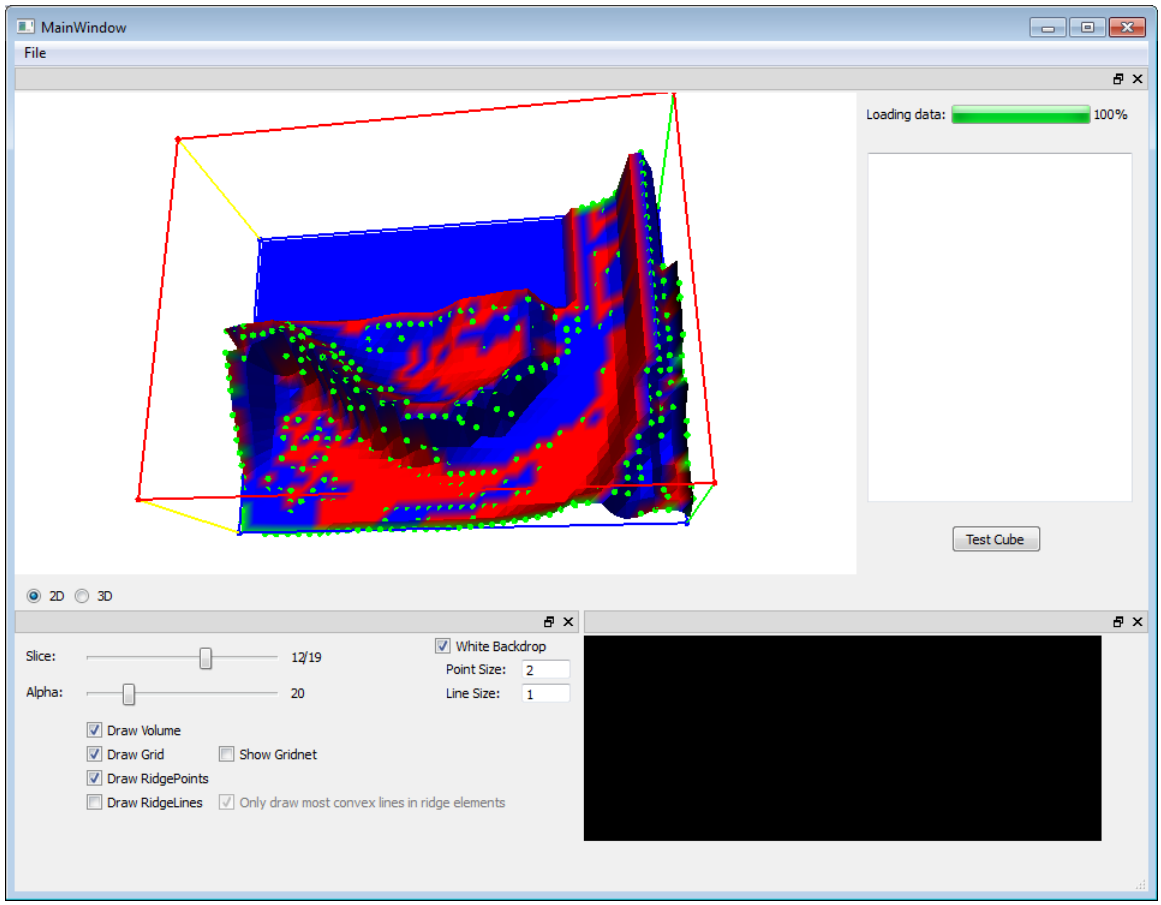


Figure 4.2: Ridge points identified on a 2D slice rendered over the volume at the appropriate height.

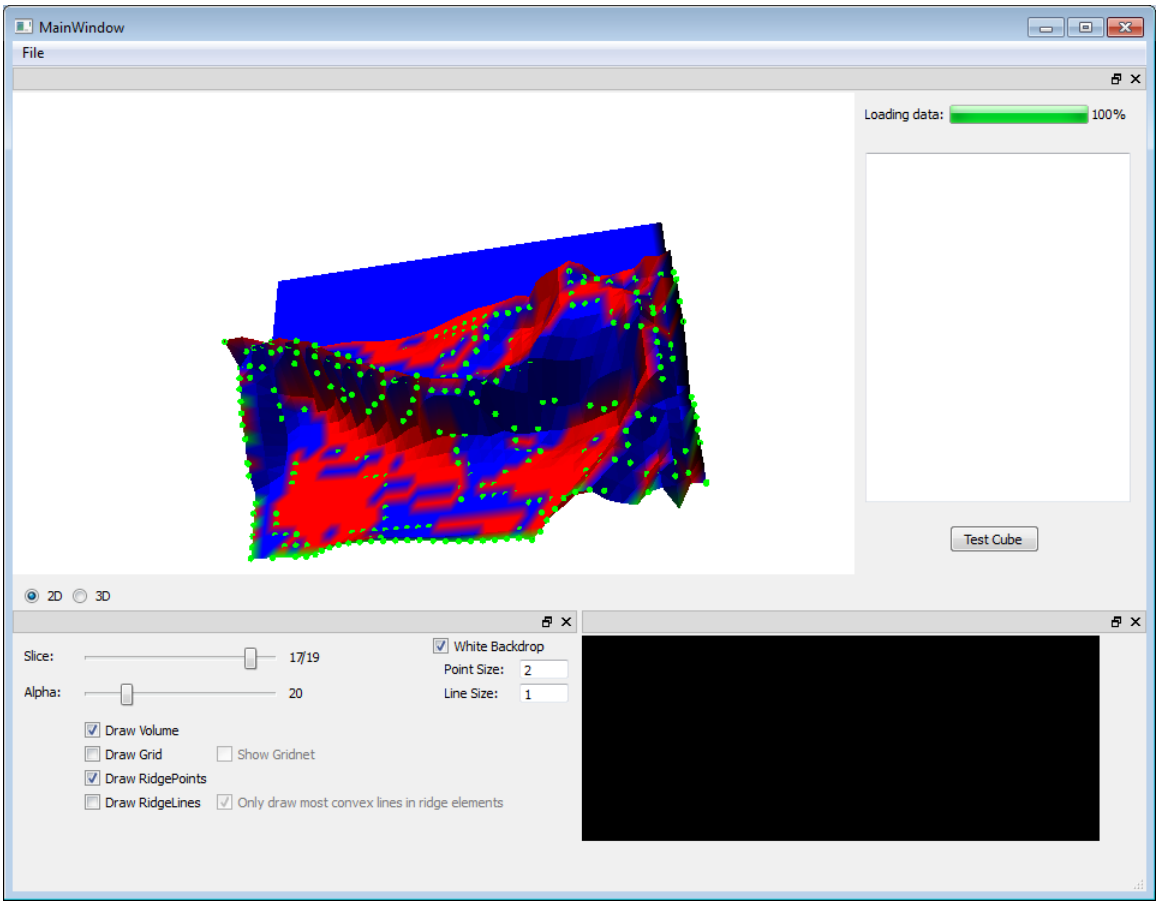


Figure 4.3: More Ridge points identified on a 2D slice rendered over the volume at the appropriate height.

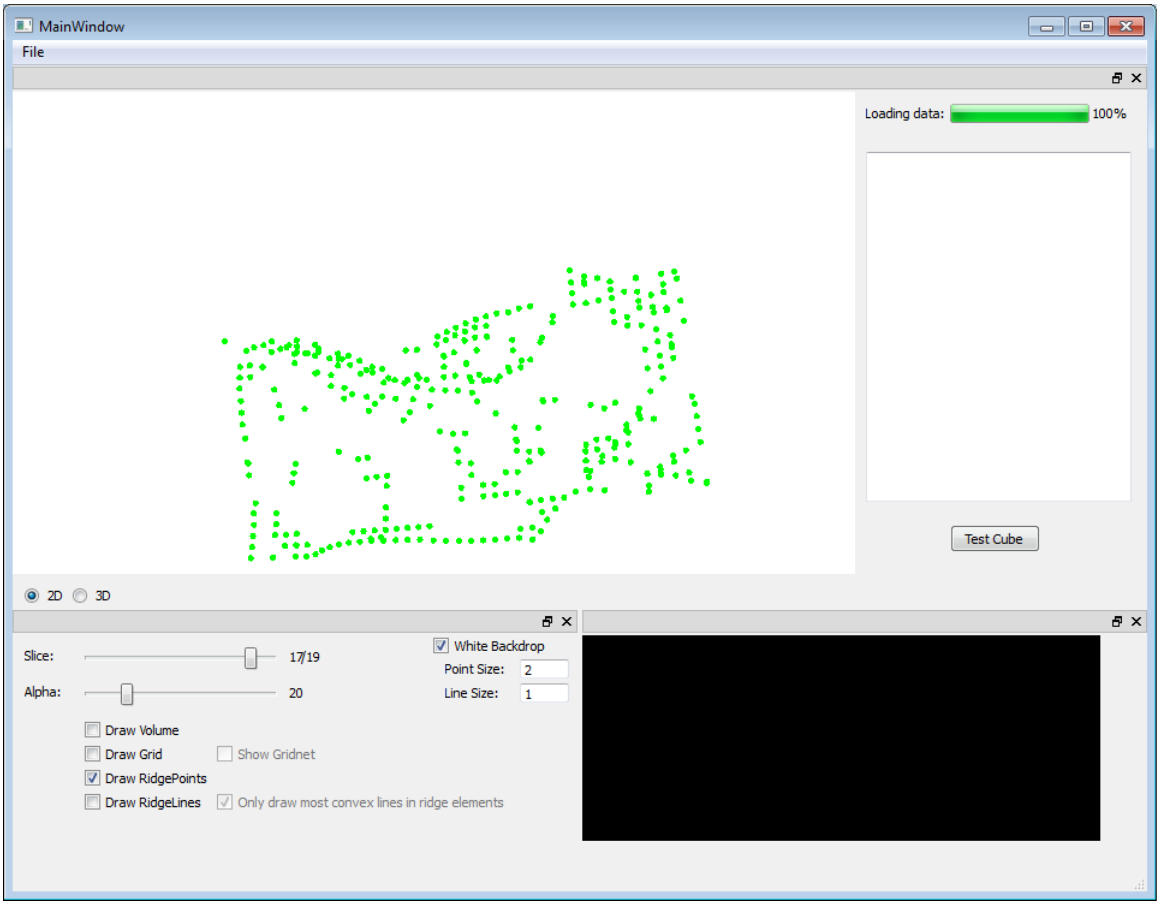


Figure 4.4: Just the identified ridge points of the slice.

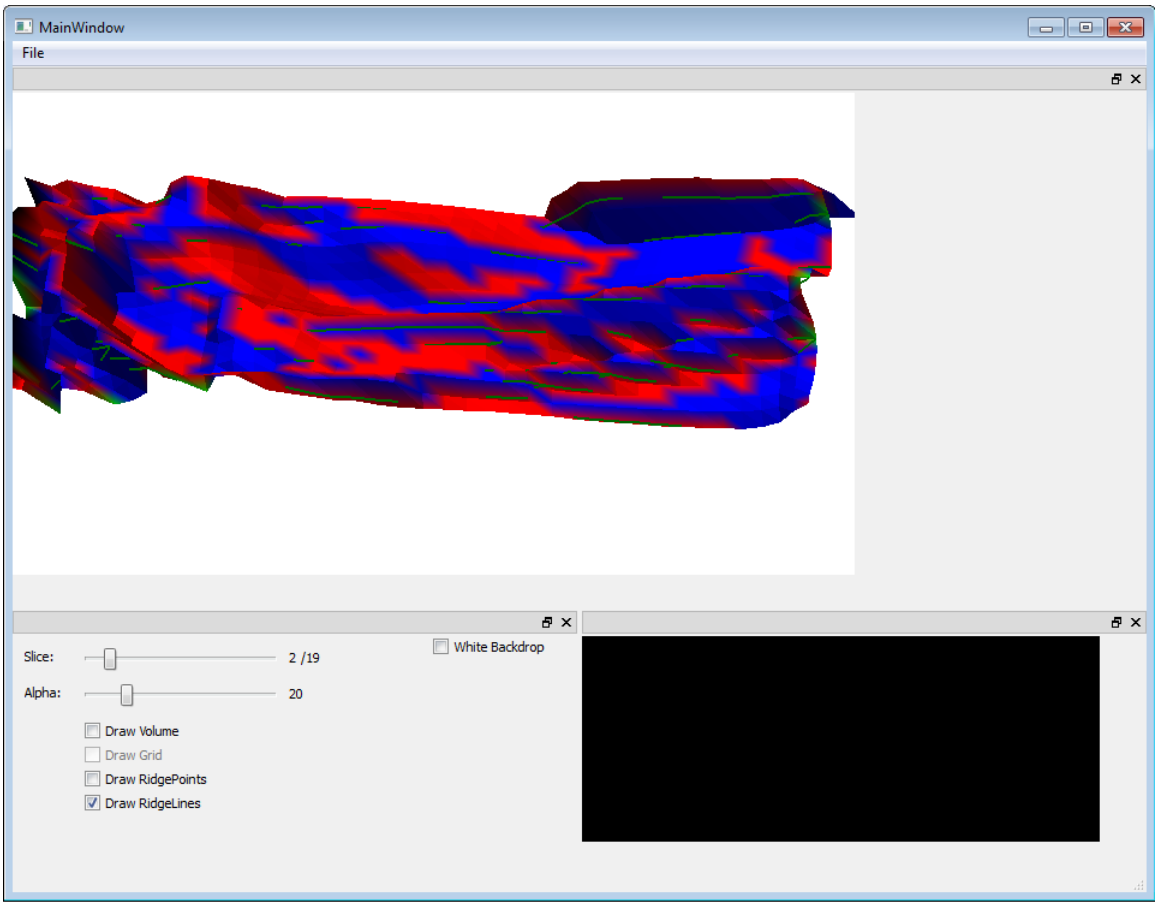


Figure 4.5: Early implementation of ridge lines.

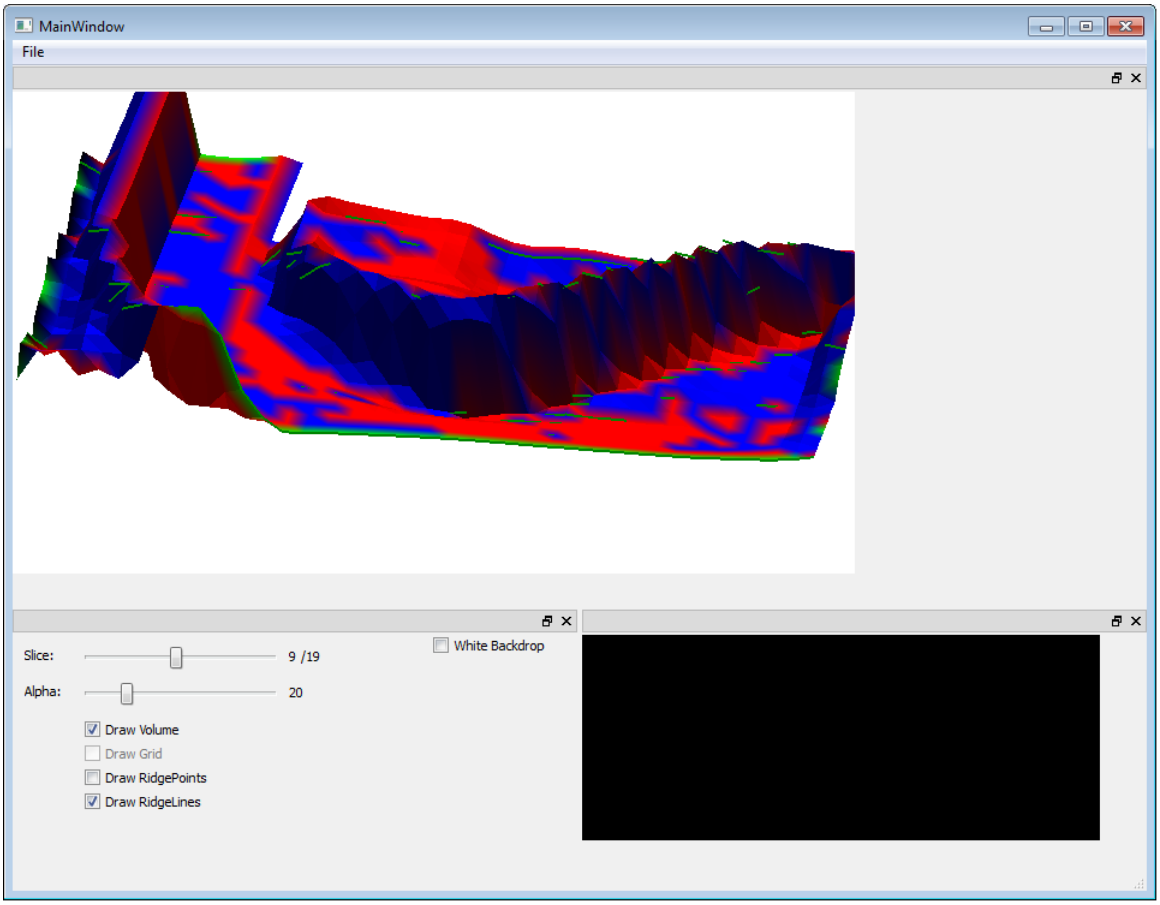


Figure 4.6: Early implementation of ridge lines.

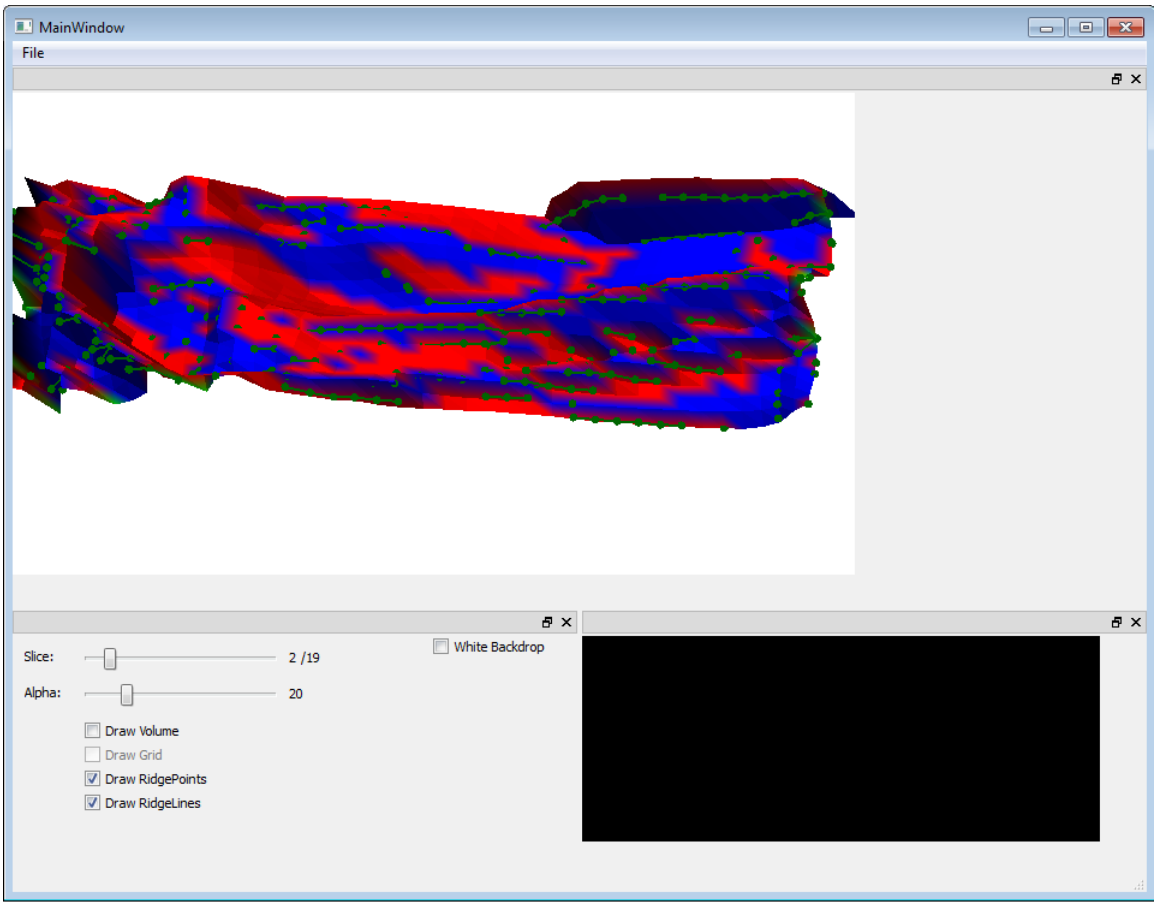


Figure 4.7: Early implementation of ridge lines with ridge points.

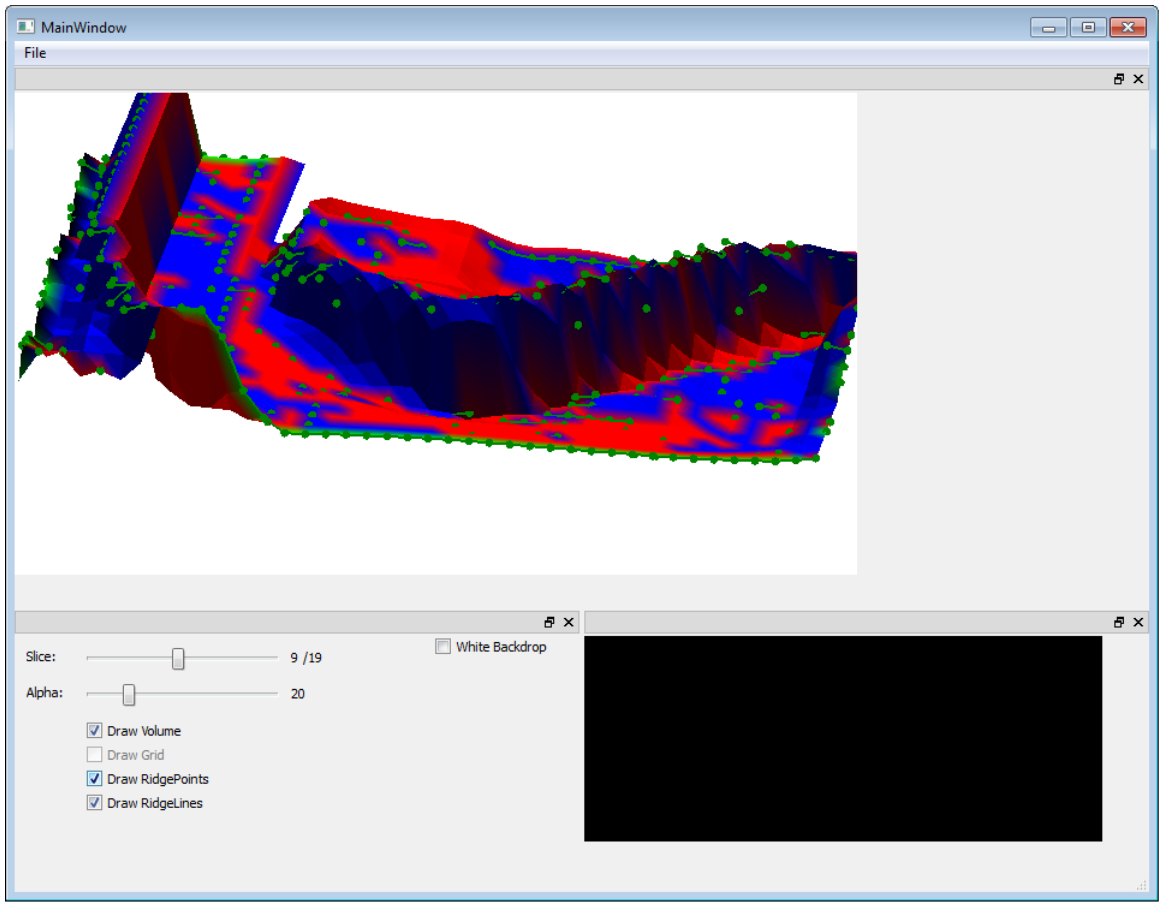


Figure 4.8: Another early implementation of ridge lines with ridge points.

4.3 The 3D Solution

In the 2D solution we ended up with a construct consisting of ridge lines, this gave us an indication of how the ridges lined up with the data as well as generating a general idea of how the different methods were performing. It is however not that helpful forming an overall picture of our construct, as well as providing a basis for our illustrative part where we need a 3D model to visualize.

We therefore expand upon the methods used in 2D to create a valid 3D model that we can then use to give us our wanted illustrative results. We are therefore expanding the methods implemented in the 2D case to give us a triangular mesh defined by the height ridges in the flow data.

The grid element in the 2D case was pretty basic, a rectangular element consisting of 4 edges. In the 3D case we are now dealing with a cube element that is defined by 12 edges. And this creates a higher complexity not just for the data needed but especially for the number of ridge points that can exist in a single grid element. We are also now describing the ridge constructs in each grid element as a set of triangles instead of a set of lines.

However we still follow the same basic steps defined in the start of this chapter but the complexity is increased since we are now dealing with the entire volume and not just a single slice. We again start with convoluting the data but here filtering the entire volume, and therefore we need a cubic filter. We go through the entire volume and calculate the gradient $G$and hessian H at each location defined as:

$$G = \begin{bmatrix} Dx \\ Dy \\ Dz \end{bmatrix} H = \begin{bmatrix} DxDx & DxDy & DxDz \\ DyDx & DyDy & DyDz \\ DzDx & DzDy & DzDz \end{bmatrix}$$

These are all identified using the sobel operators defined below. These values are then stored in each data point for quick access when needed in later calculations.

Some example sobel filters used for 3D:

$$DxTop = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} DxMiddle = \begin{bmatrix} -2 & 0 & 2 \\ -4 & 0 & 4 \\ -2 & 0 & 2 \end{bmatrix} DxBottom = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$DzDzTop = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix} DzDzMiddle = \begin{bmatrix} 2 & 4 & 2 \\ -4 & -8 & -4 \\ 2 & 4 & 2 \end{bmatrix} DzDzBottom = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix}$$

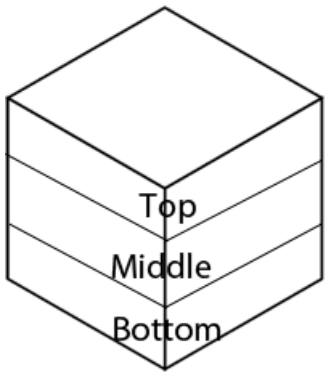


Figure 4.9: The cube filter designed in 3 layers, top middle and bottom.

As we can clearly see here the complexity of the convolution and the amount of data it produces is greatly increased, but the end results are very much the same. We still end up with a gradient and a hessian defined at each data point that we can use to identify possible ridge points just as in the 2D case. So in the end we now use a substantial more data, and it therefore takes a bit longer but since we are keeping the main principle of convoluting the data using a simple filter the complexity is kept down and we can still follow and fulfill the steps required to define a suitable model for our illustrative results.

The next step in the process is the eigen analysis. We now have more eigenvalues to consider but are still only looking for the smallest one so that we can get the eigenvector that corresponds to this smallest value. This results in more or less the same procedure as in the 2D case but with a bit more work done, much the same as with finding the derivatives of the data.

So far we have seen that with expanding the implementation in dimensions we have still kept the basic principles of our solution and only the magnitude of the data has been affected, the end results are more or less the same as we still need the same values to identify possible ridge points. We are still looking for the same type of ridges, and we are essentially still checking edges in the end - but the complexity of the 3D implementation comes in the form of creating the resulting model.

In the lower dimensional case it was just a matter of connecting points to form lines, that then formed a resulting image defining the height ridges in a 2D slice. In 3D we need to define a set of triangles that not only represents the different formations of the ridge points within the edges of a cube but also makes sense in the resulting model. There are many ways to approach this, and as we showed in our 2D case we based our solution on the marching ridges algorithm, and this was in turn based on the more well known marching cubes algorithm. [**?**]

Now in the marching cubes algorithm there are a set amount of cases that can occur within a 3D model and these cases are then reduced into 15 unique triangle formations that are rotated and moved around to form all the possible arrangements. These different cases are represented using an edge list that defines the many cases and is stored in a list that is checked when inspecting a cube element when executing the algorithm.

We did try this at first, using a predefined edge list that would define our grid elements and then give us the resulting correct triangle arrangement, but after further examining the data we have we found that the cases used in the marching cubes algorithm did not correspond to the different cases we had because of how the ridge points are detected they create different patterns that are harder to define using a set list of unique edges. This resulted in that we had to then identify the edges that have a valid ridge point, and construct the triangles as simply and as straightforward as possible without ending up with artifacts or clutter from

wrongly arranged triangles.

But first we have to find the ridge points needed on each edge of the grid element so that we can identify the different cases and assign appropriate visual representation in form of triangles.

This is done in a similar approach to the 2D case where we go through each grid element and check all the edges individually for a possible ridge point using the values we have obtained through convolution and eigen analysis.

We again start with identifying and checking the transverse direction of the edge points now using the 3x3 matrix C using the same definition as in the 2D case. Similar to the eigen analysis we now have an added eigenvector we need to consider, but the end result is still the same - in that we are still just after the greatest eigenvector from C so that we can compare the edges transverse direction with the average transverse direction and make sure it is not negative.

We now move on to detecting a zero crossing in the first derivatives on of the edge by interpolation, and if a zero crossing is found we have a possible ridge point that is affirmed by the second derivatives being negative in the same way as we did in the 2D case.

So as we can see identifying the ridge point in the 3D case is not much different from the 2D case, but as mentioned earlier there is a bigger change in the last step of our implementation as we now need to define the triangles defined by these ridge points and get a resulting surface that we can use as a model.

As we have been focusing on keeping the approaches straightforward and simple we will continue along these lines when constructing the triangles in the grid elements. We therefore take the grid elements in a case by case basis and check exactly how many ridge points are contained within them and construct the most straightforward triangle orientation that can be found without generating bad or conflicting results. The grid elements with exactly 3 ridge points are in this case trivial and are drawn as a single triangle, and the rest of the triangles are treated in an equal manner using this algorithm:

In principle the algorithm designed reduces the problem of constructing an interconnecting and overall model into defining the triangles in each grid element separately. Because there are so many possibilities as well as the complexity of the data that identifies ridge points creating a consistent list that represents all would

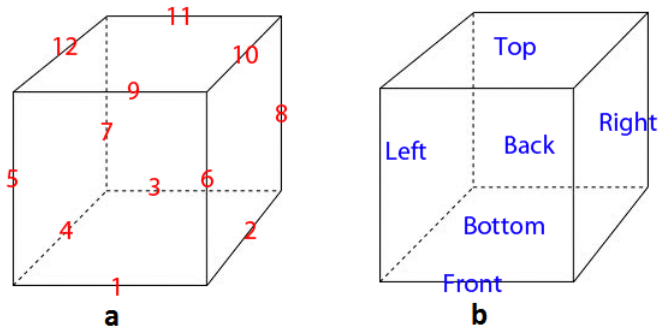


Figure 4.10: A grid element defined as a cube. The edges and faces on the cube help define and organize the grid element so that we can construct our triangles. a) shows the 12 edges that define it b) shows the 6 faces that are used to check adjacent and opposing edges for ridge points.

not be the optimal way to approach this problem. Instead we take the fact that only a single ridge point can exist on each edge as well as the fact that we want simple and well formed triangles that are not obscured or fold in an unnatural way. This is done by organizing the 12 edges into 6 faces that define the neighborhood of the grid element. This is shown in this figure4.10.

We then use adjacency lists and define the triangles in a consistent manner for each grid element. We want to combine the closest ridge points first, so opposing faces are deprioritized. In this way we can separate the grid elements into 2 different cases. The first case consists of triangles formed by adjacent faces and the second case consists of triangles formed by opposing faces. So in essence what we do is:

1. Check if there are lines(defined as two separate ridge points) in the cube with shared vertices and connect them.

2. Prefer adjacent faces.

3. If no adjacent lines are available look for opposing lines to connect.

Where we define neighboring faces as faces that are directly in contact, so for instance the front face would have four adjacent faces in left,right,top, and bottom. Opposing faces are then defined as faces that are not directly in contact e.g. the front and back face. So if we look at the figure **??** we can see that in the case

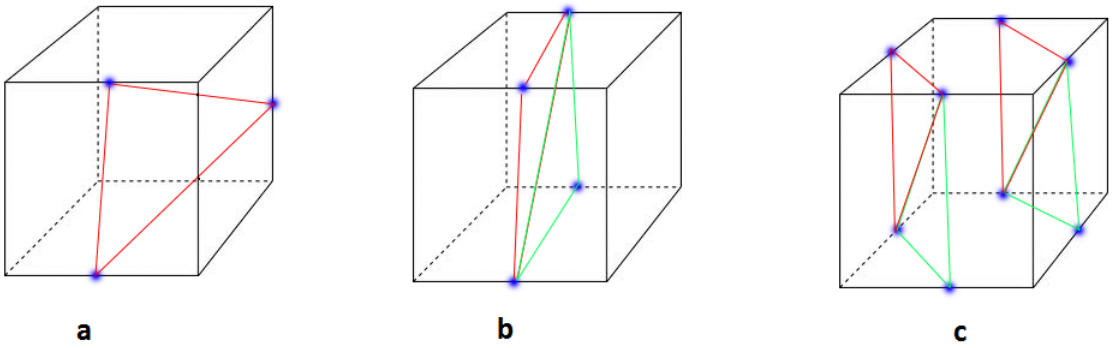


Figure 4.11: A selection of triangle constructions that can be found in our grid elements. a) the trivial solution b) the neighboring solution c) a more complex neighboring solution with a predefined configuration.

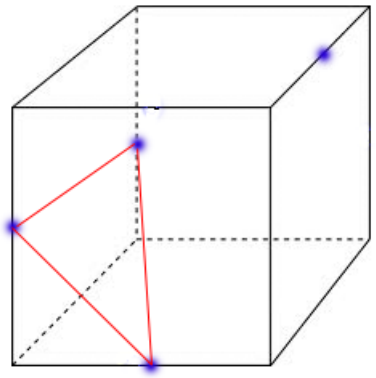


Figure 4.12: Another special construct consisting of 4 ridge points but only a single triangle.

of a) we simply connect the shared lines to form a triangle even though we don't have more than 1 shared line. In the event of b) we first connect the adjacent lines formed by the red line where lines on both the front and top face share vertices. Then we see that the there are also adjacent lines at both the bottom and back faces and therefore connect them to create our final structure. In our last example we have a bit more of a tricky situation seeing as there can be multiple final structures depending on how we start and how we draw the lines between the points. In these special cases we simply define a solution and when it is identified we draw it the

same way each time. This is done by how we implement the priority of faces so in this case we check the left side before the right side for a shared line and therefore end up with the configuration shown. Another special case that can be found is the instance of e.g. four ridge points found where three of the points are adjacent and the last point is on an opposing face. This case is shown in this figure 4.12 and our solution is to simply ignore the point that does not share a line with any of the other to prevent clutter and to keep the constructs simple. After we created the model we needed to make sure it was valid and working as intended. We therefore exported the resulting mesh and had a look in meshlab to get a better understanding on how the resulting model looked. However to end this section we round off with some images taken from the triangles created in our solution.

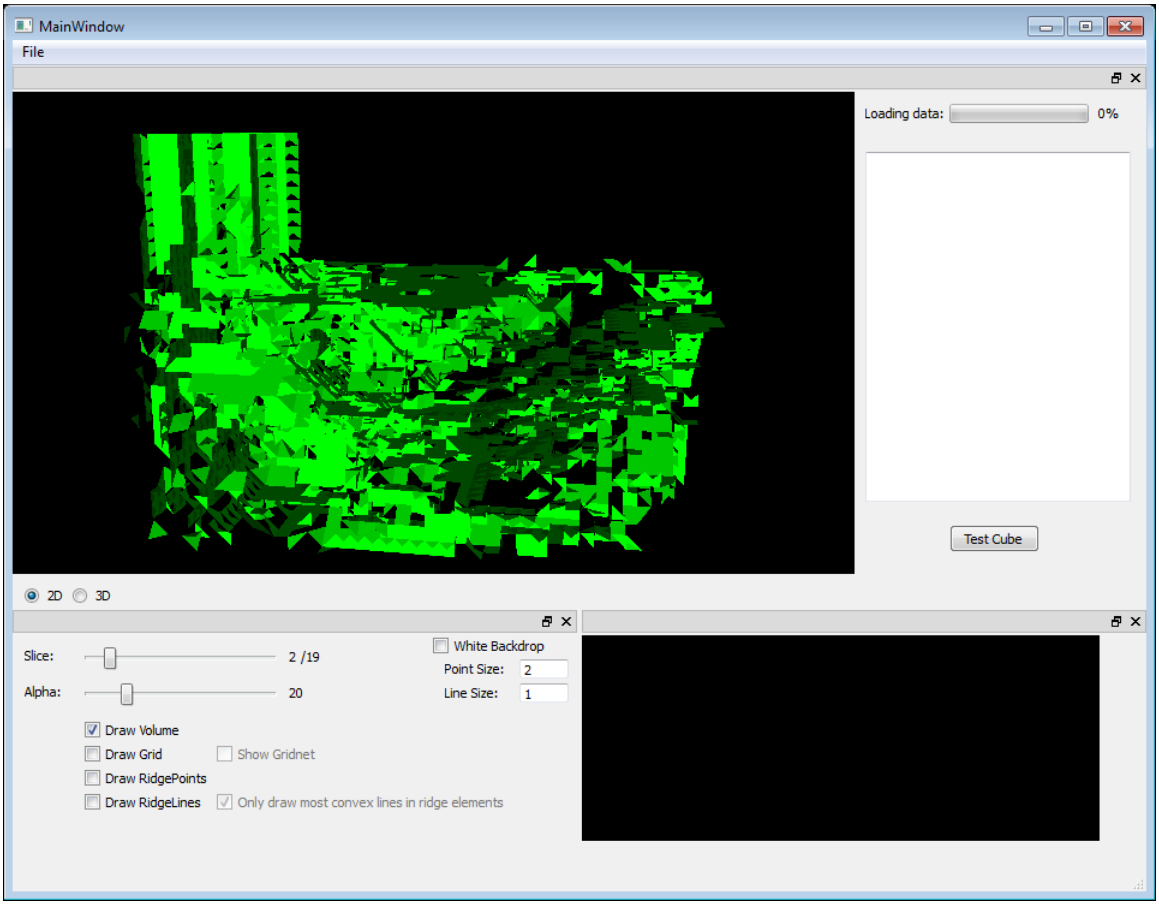


Figure 4.13: Triangle mesh formed from our 3D implementation.

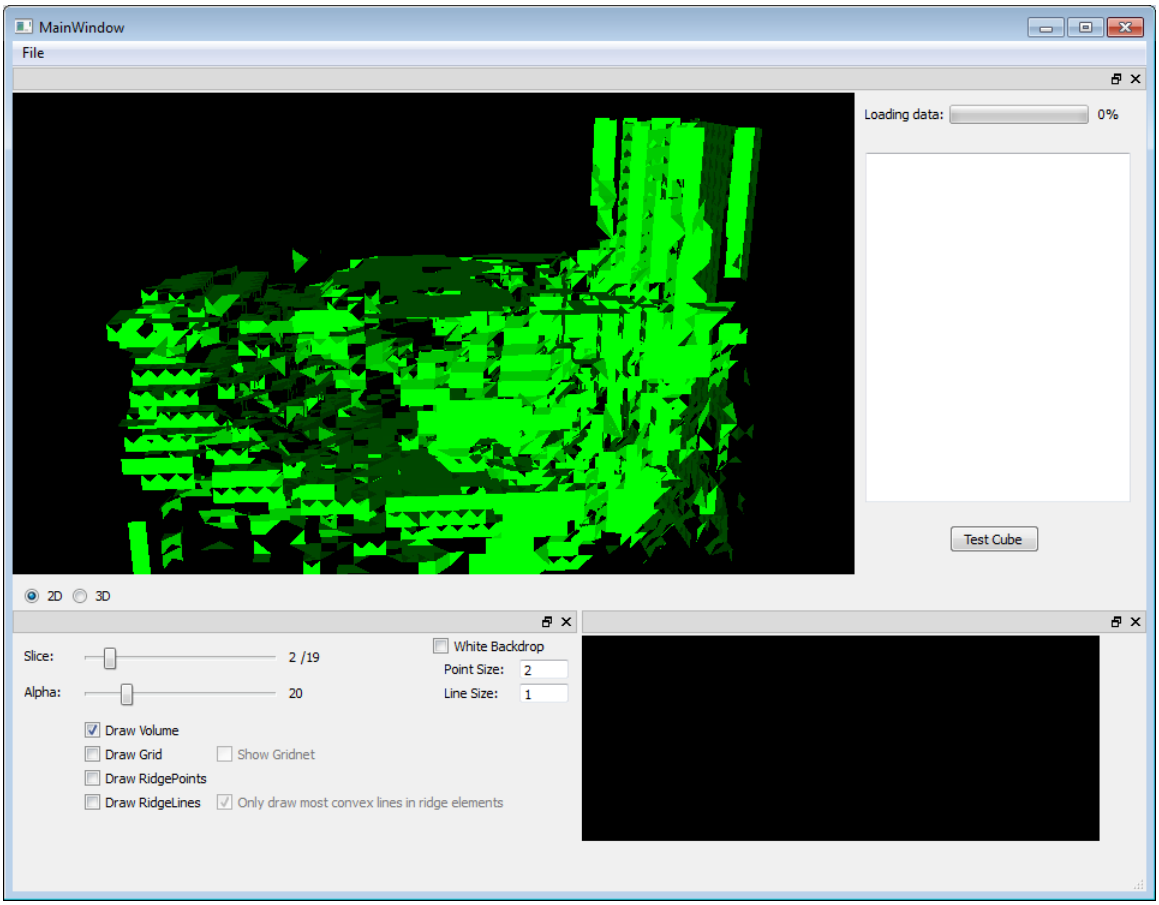


Figure 4.14: A closer look at our triangle mesh formed from our 3D implementation.

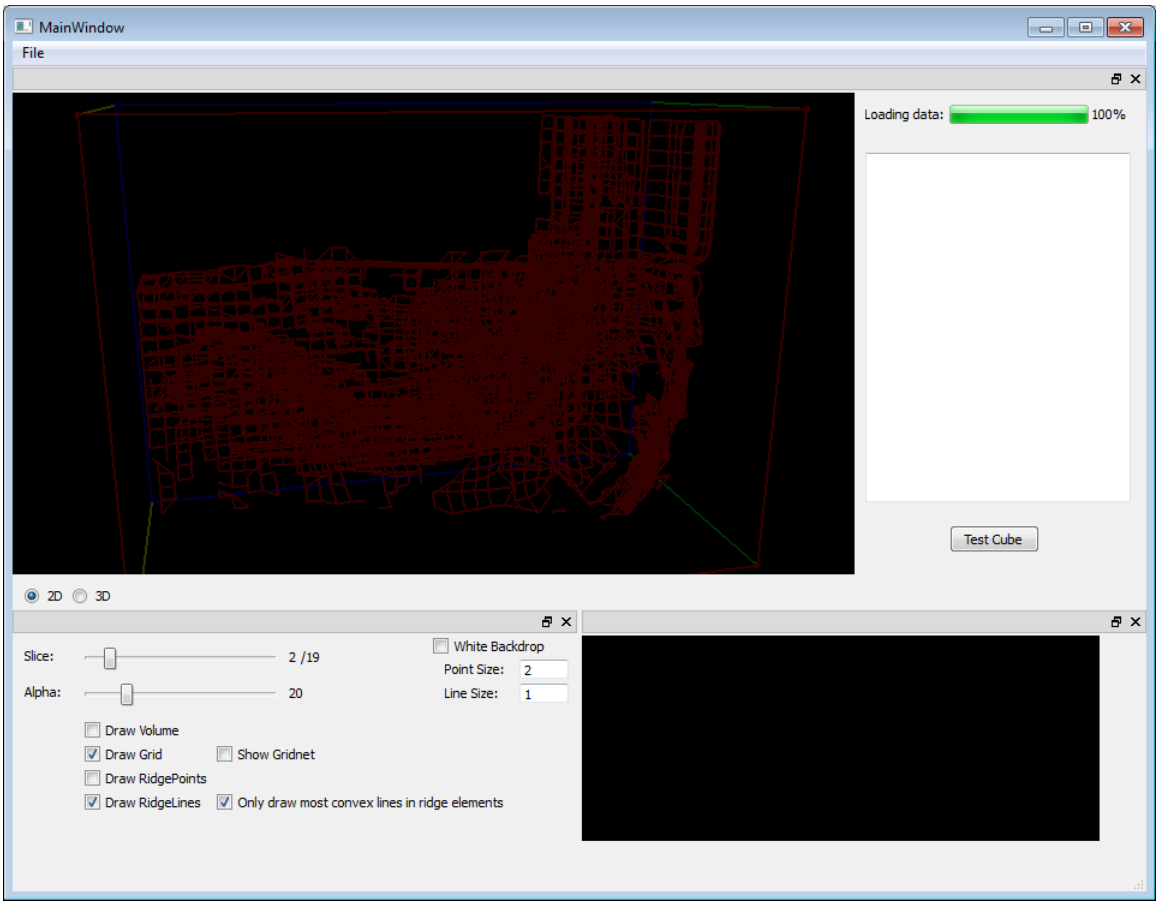


Figure 4.15: Ridge lines as they are defined in the grid elements combined to form a skeleton of our structure.

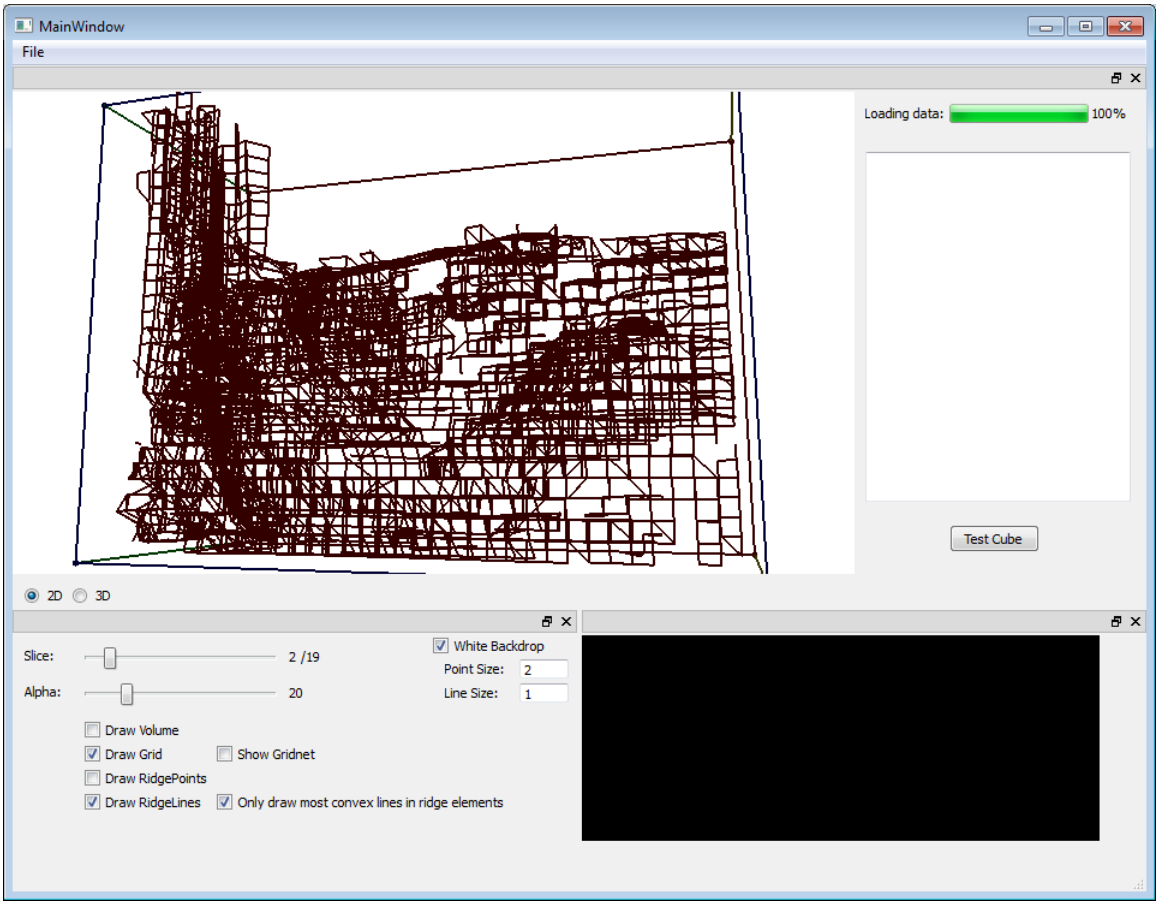


Figure 4.16: Another view of the ridge lines

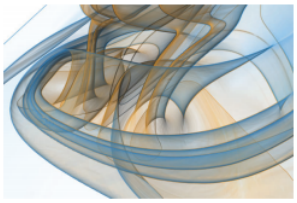


Figure 4.17: A normal variation view-dependent transparency rendering. From [16]

The Illustrative Approach

To create our illustrative result we need a custom built shader. We base our approach on some of the techniques utilized in the paper by Hummel et al. [16] Specifically we intend to implement a shader that utilizes a normal variation to create transparency so that we can more easily see what is going on within our structure. This approach can be seen in fig. 4.17

So what is needed is a shader that can process our model in such a way that it creates a normal variation view-dependant transparency of the LCS created by the ridges previously extracted and combined. We therefore need to make this a two-step process seeing as we have to take into account the fact that it is view dependant. This means that we need to create a layered approach to our shader in such a way that we can strip it down and show the different structures in the data.

CHAPTER

5 Results

In the end, it's not the years in your life that count. It's the life in your years.

Abraham Lincoln

The end results in this thesis prove the complexities and difficulties inherit in both flow data and how to visualize them. Having provided an accessible way to extract and create the needed structure to visualize using mathematical procedurs and finding both normals and we have a brief look at how the end results came together to form a LCS created by reducing the data into separate gridelements and then producing a compatible triangle mesh that defined the inherent structures within the flow.

By applying the implemented methods on our testdata sets we have created several triangle meshes that represent the LCS within the complex flowdata. We utilized our ridge extraction method to generate and verify ridgepoints and by basing our surface constructing method on existing techniques such as the marching ridges [6] as well as other important ridge generating methods like filtered AMR ridge extraction [34] and Parallel Vectors [29] we constructed a resulting model in both 2D and 3D that can then be used or exported in such a way that we can inhance and focus on its inherent properties. The method is also easily expandable if we take into account that the FTLE values are predefined and the datasets are

managed by a cartesian grid.

We were however not able to integrate our custom shader to provide an illustrative approach to this data due to lack of time and the inherent complexities in handeling the data as well as definig a proper solution. The shader proved more complex than previously thought and nfortunately the end results of our proposed solution were not met in the end.

There has however been provided some shaded visualizations of our final aquired 3D model of the testdataset taken in two different time intervals.

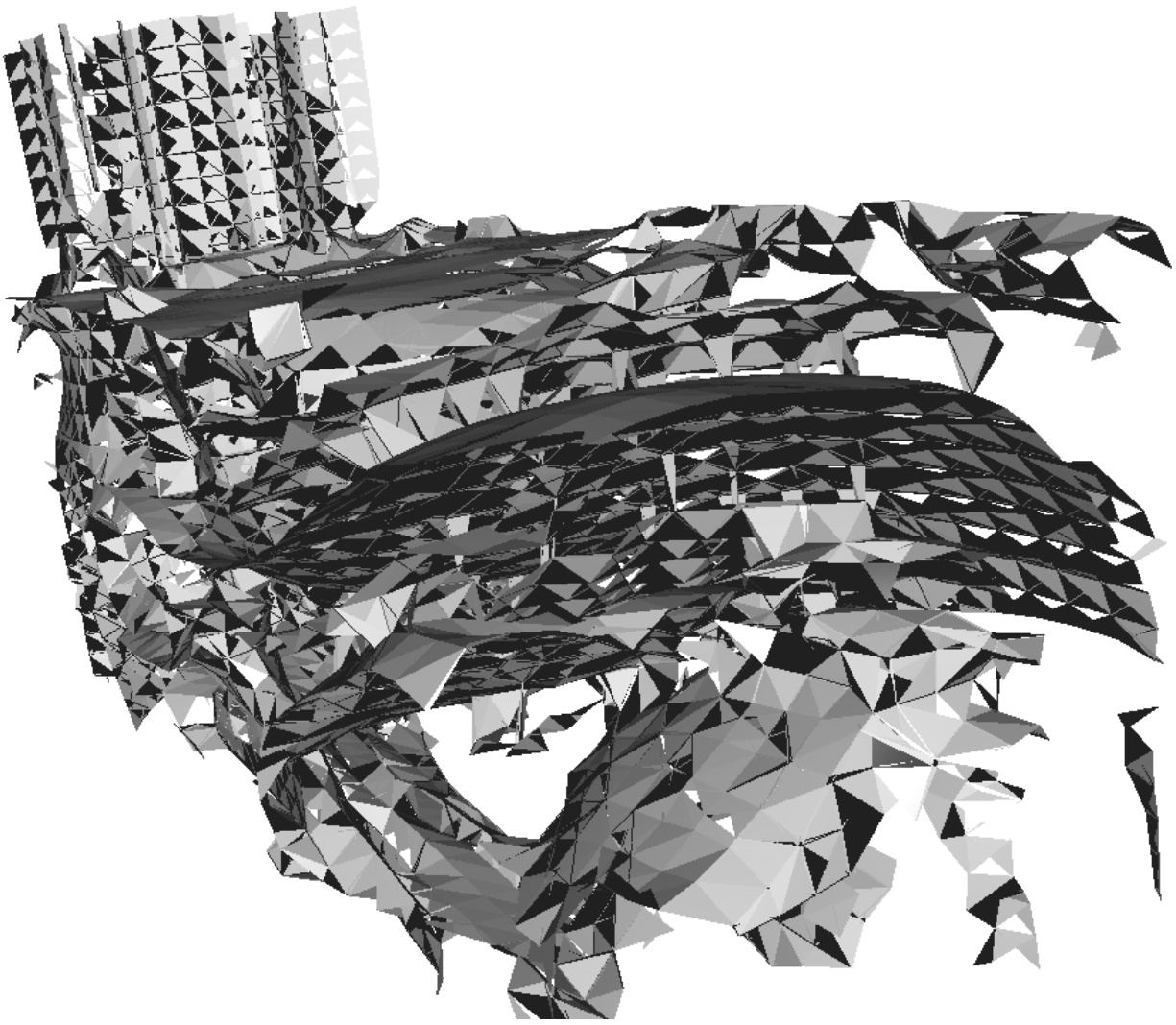


Figure 5.1: Resulting triangle mesh with simple phong shading of our testset, TS43.

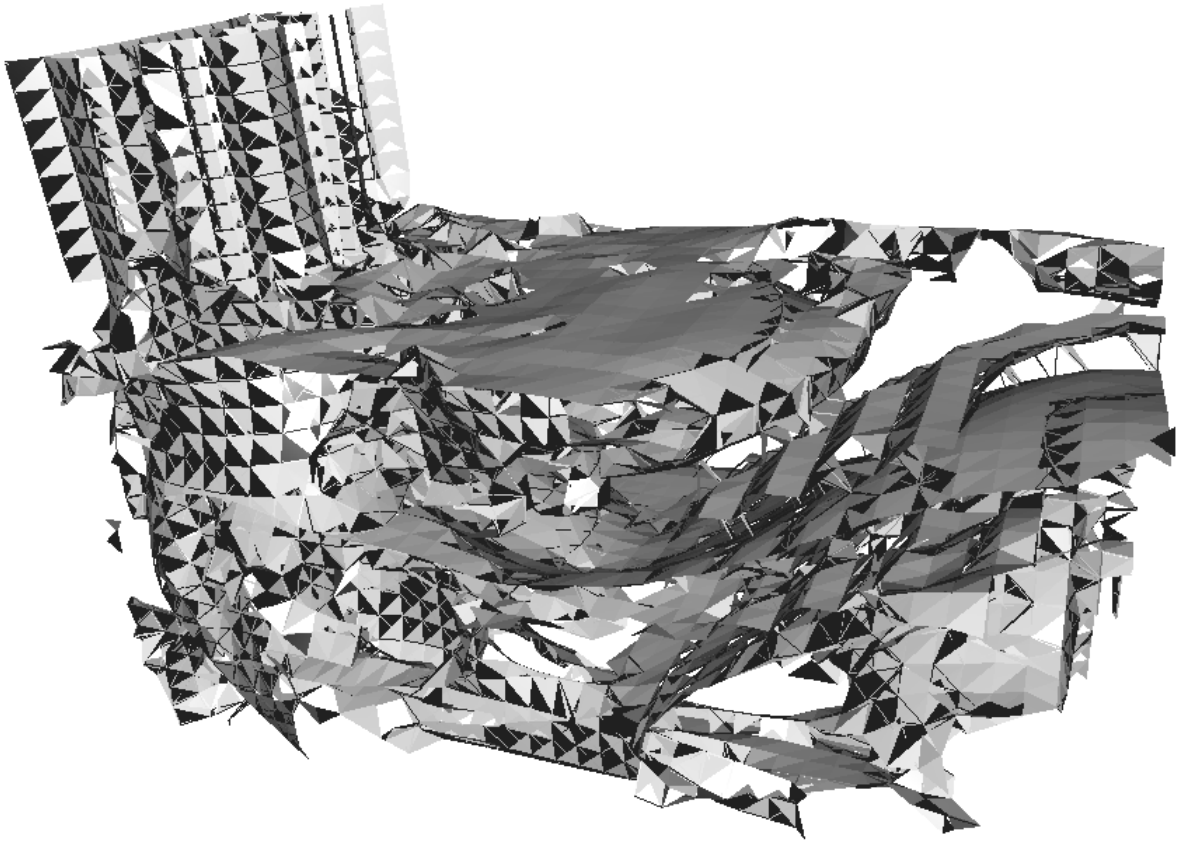


Figure 5.2: Resulting triangle mesh with simple phong shading of our testset, TS43.

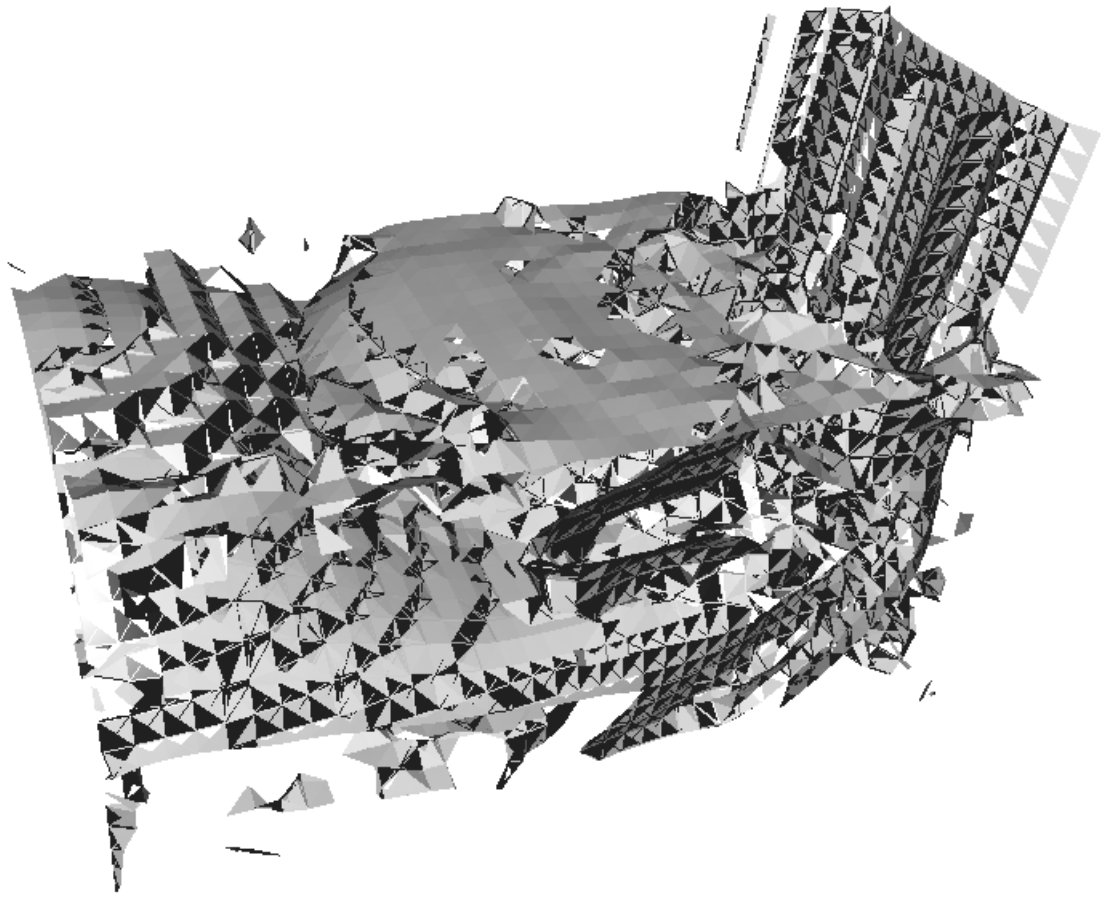


Figure 5.3: Resulting triangle mesh with simple phong shading of our testset, TS43.

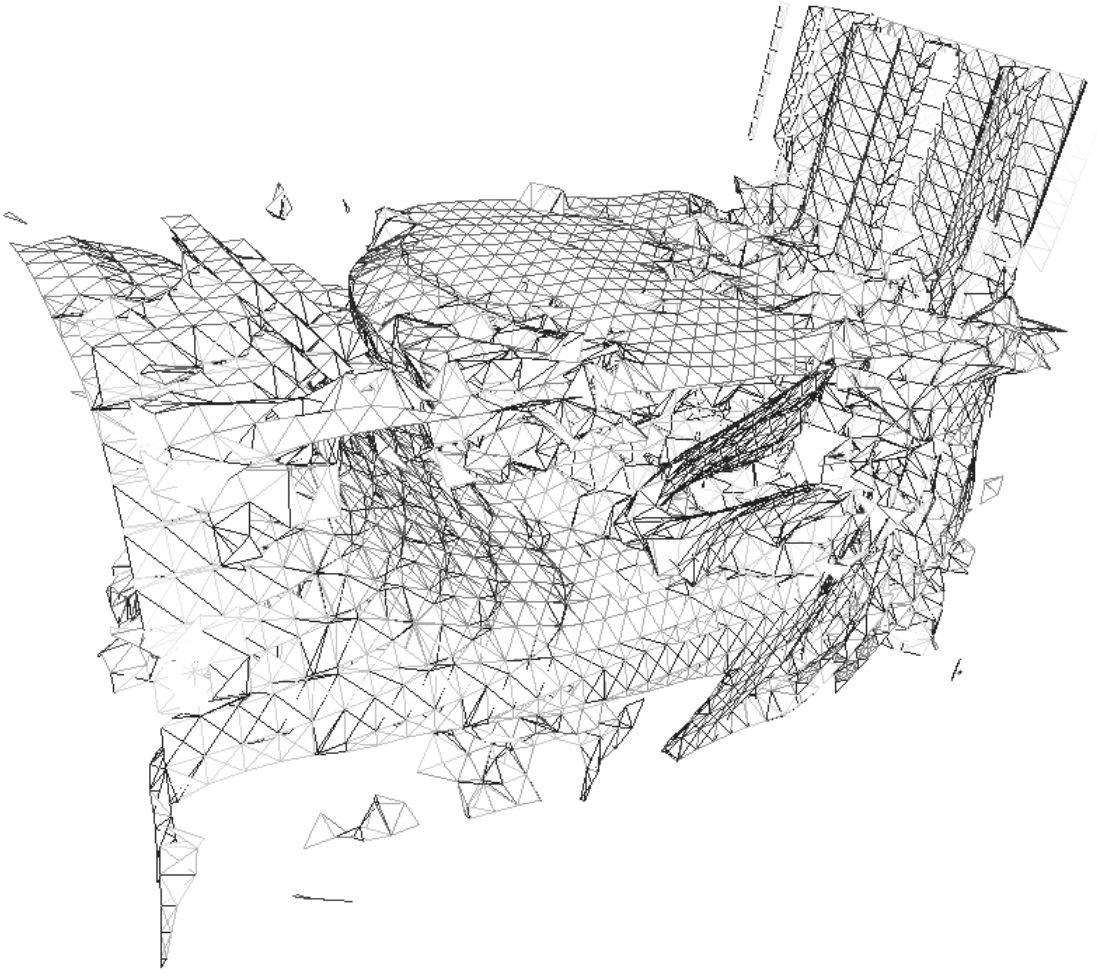


Figure 5.4: Resulting triangle mesh with simple phong shading of our testset, TS43.

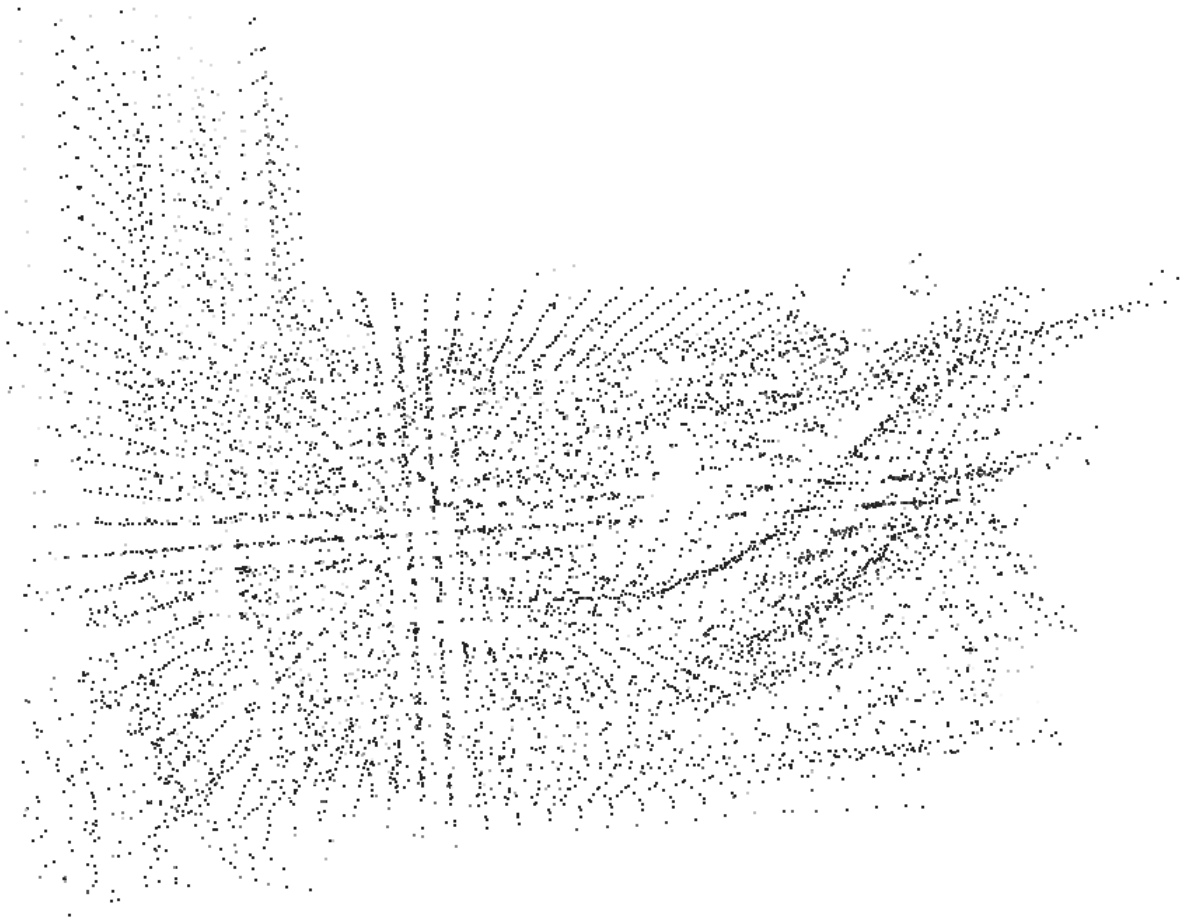


Figure 5.5: Resulting triangle mesh with simple phong shading of our testset, TS43.

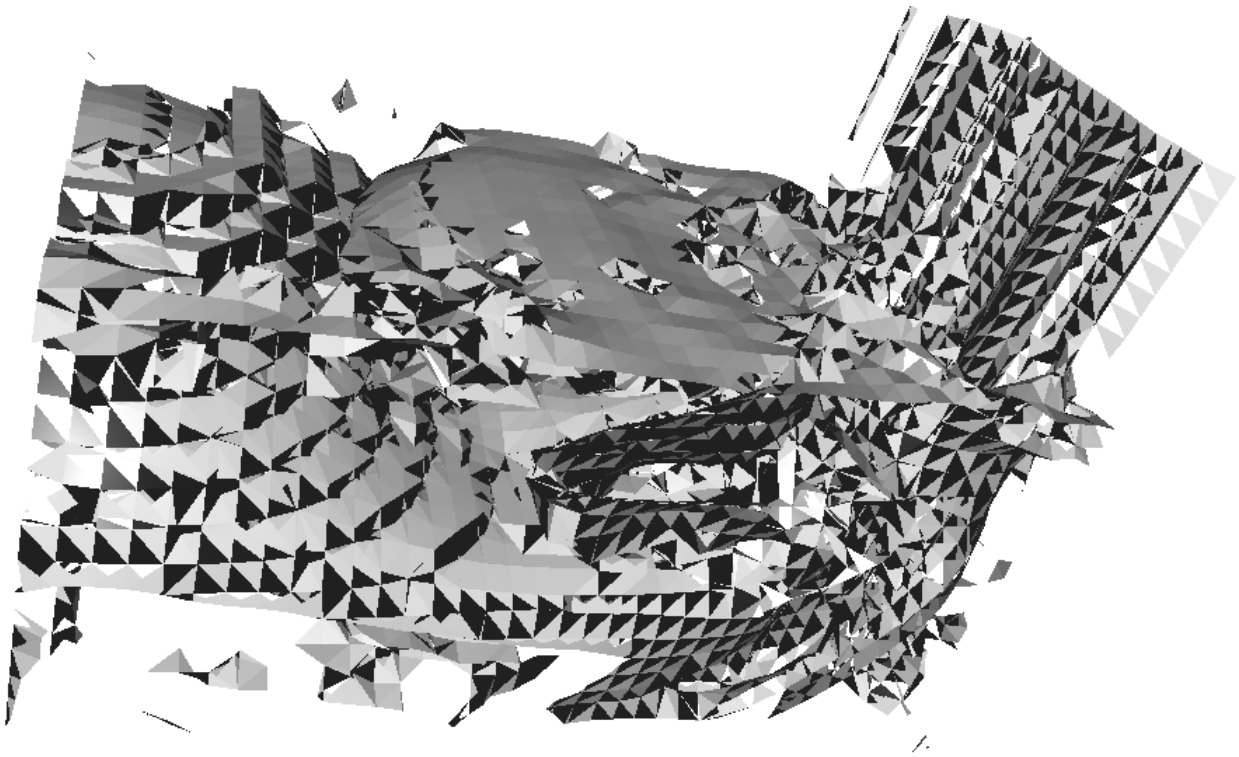


Figure 5.6: Resulting triangle mesh with simple phong shading of our testset, TS43.

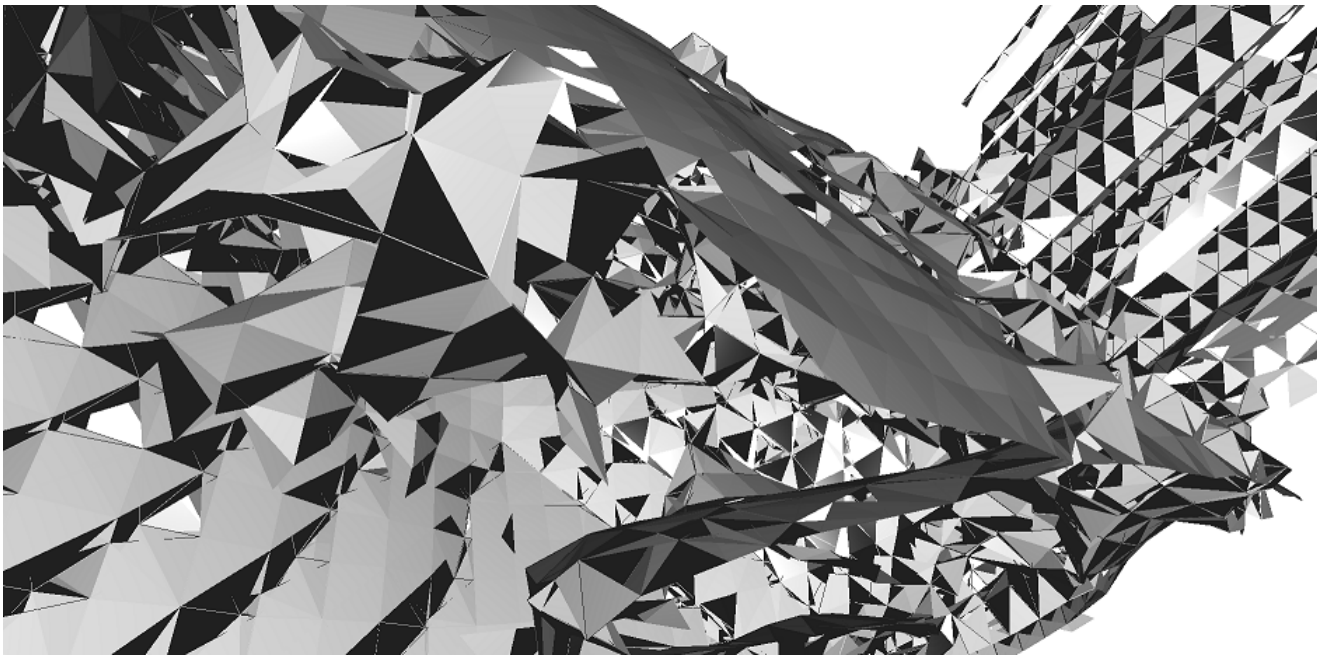


Figure 5.7: Resulting triangle mesh with simple phong shading of our testset, TS35.

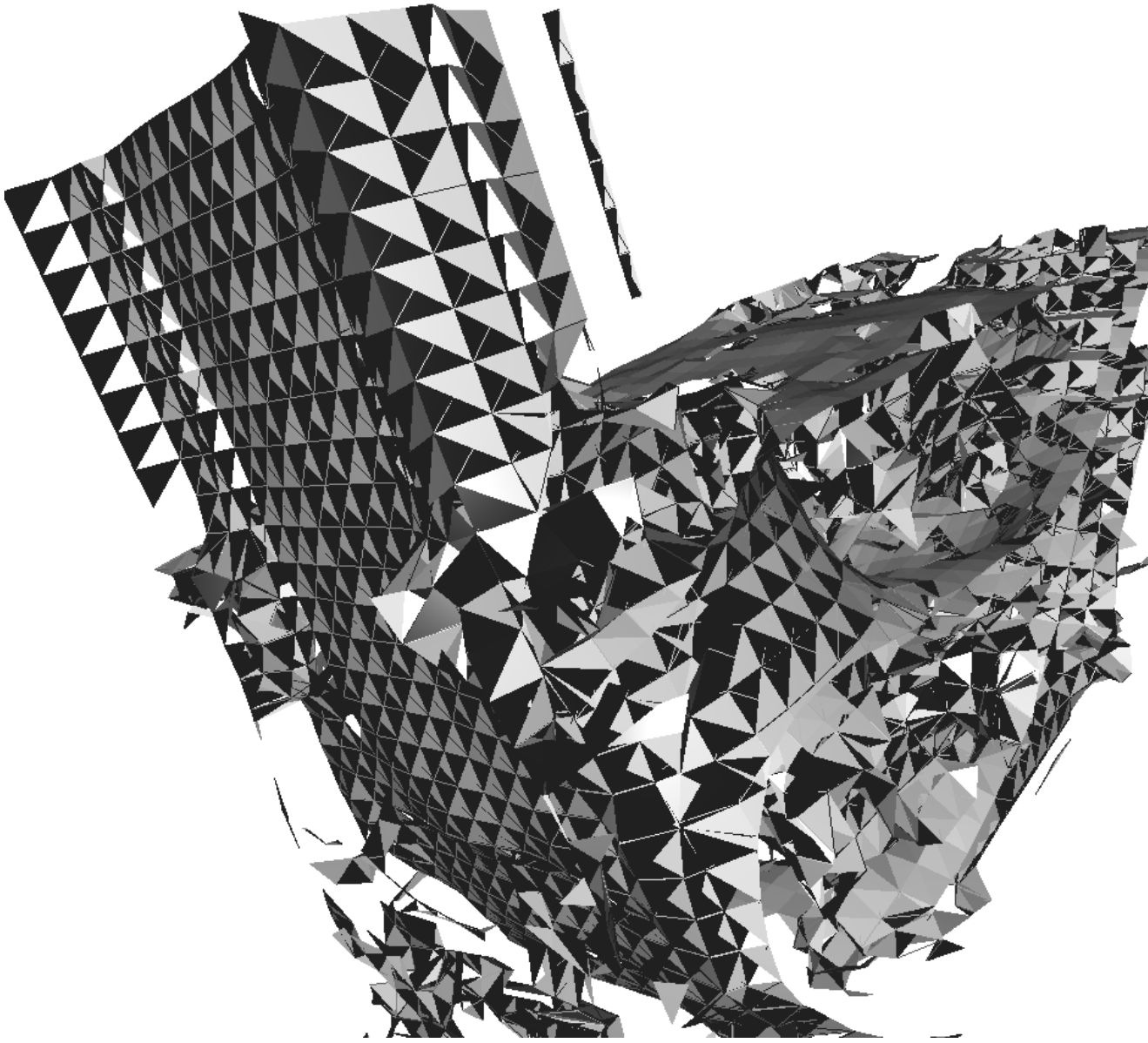


Figure 5.8: Resulting triangle mesh with simple phong shading of our testset, TS35.

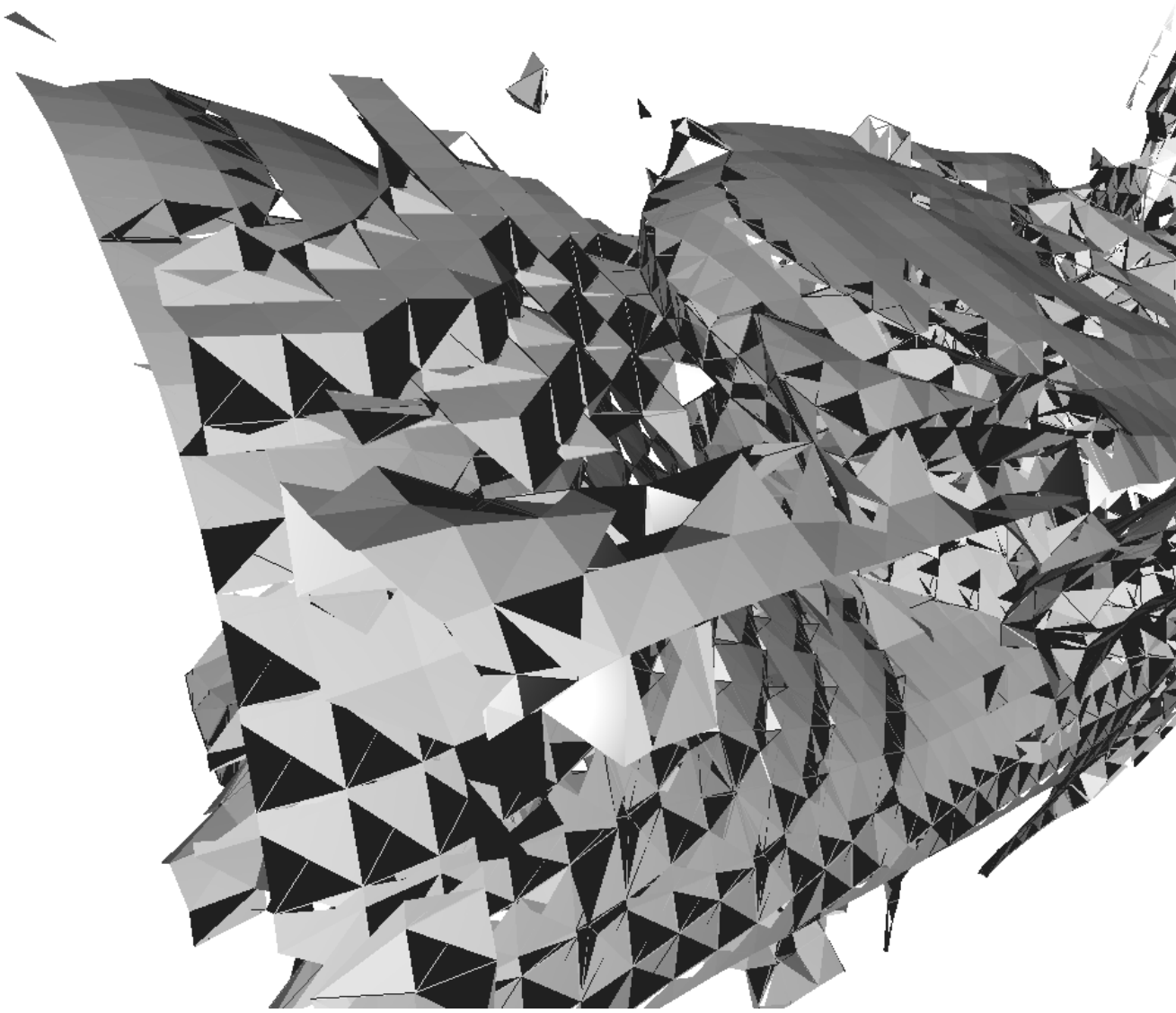


Figure 5.9: Resulting triangle mesh with simple phong shading of our testset, TS35.

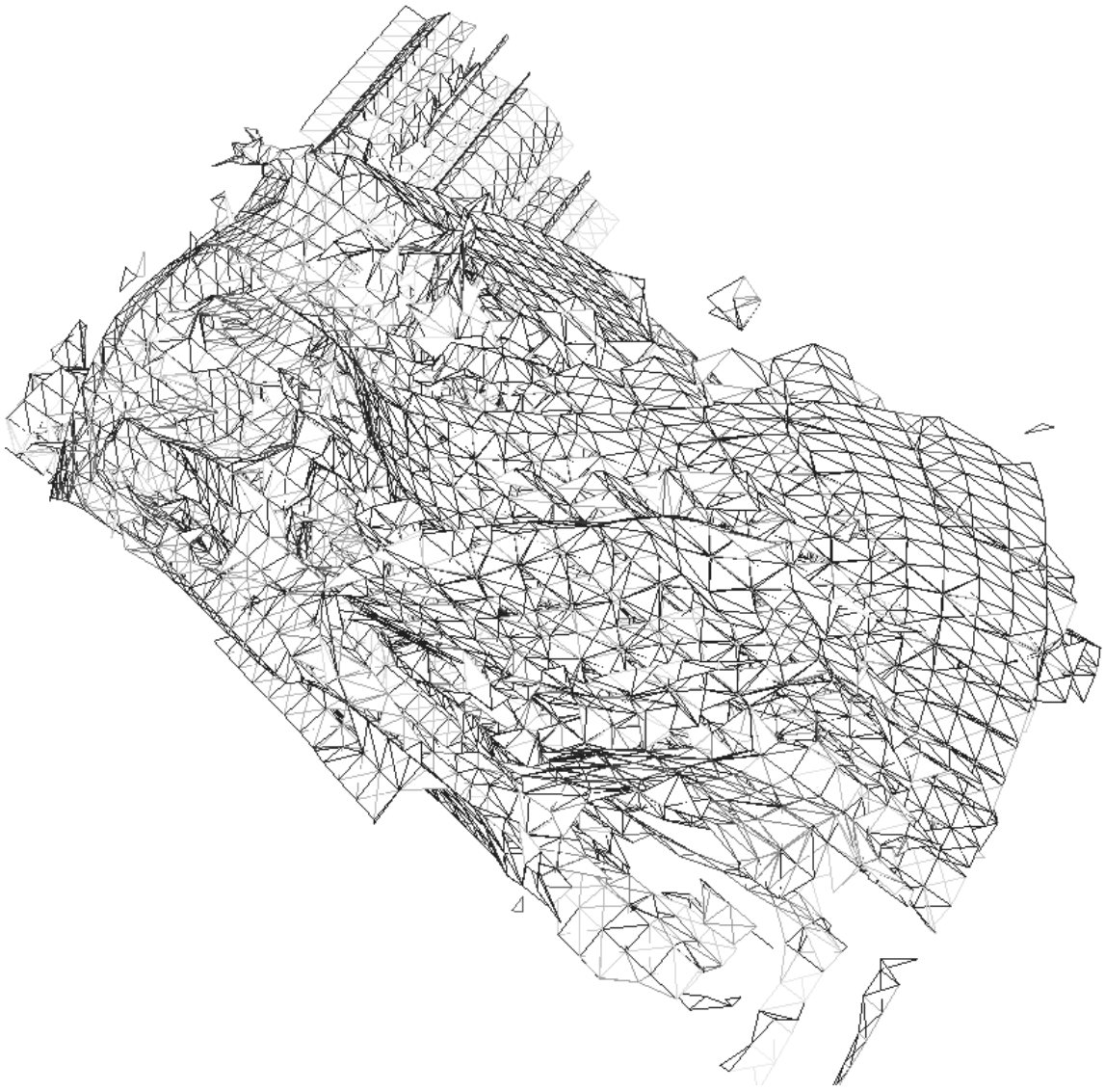


Figure 5.10: Resulting triangle mesh with simple phong shading of our testset, TS35.

CHAPTER

6 Conclusion

When I examine myself and my methods of thought, I come to the conclusion that the gift of fantasy has meant more to me than any talent for abstract, positive thinking.

Albert Einstein

Summary and Problems Faced

To conclude the project there are a lot of possibilities still within this area. And it is important to notice the complexity and the size of even such a simplistic and well defined approach. Having success in preparing and identifying interesting areas of visual output as well as the vast Creating visually stimulating shaders and utilizing different techniques taken from both illustrative visualization and other areas could prove very interesting with this type of data. As we can clearly see in the paper by Hummel et al. [16] there is a clear potential within this area.

However handling such an amount of complex data proved more difficult than early assumptions as well as the complexities going from a lower 2D approach and extending it to a third dimension also had its difficulties. In the end we can conclude that there is definitely interesting aspects if further explored within this

area and the fact that complexity is one of the major issues within the field of flow visualization it is going to be very interesting to see what the future holds both in form of having more interaction as well as being able to handle the data better. Extracting and finding the necessary information isn't always straightforward when working with this type of data and that was shown in this thesis.

The solution we went in with was sound and although we had to try quite a few approaches to extracting and defining the data in 3D, mostly because the preexisting methods and algorithms usually does not have all the information gathered as well as using more of a seeded approach to constructing the necessary structures. it was in the end implemented as we had intended in regards to the extraction and forming of the final model.

Future Work

As mentioned there is a huge potential for future work within this area, not only to further test the boundaries and try to reduce the complexity, but also in that we try to create easier and simpler implementations that still work fast. An illustrative approach to this area can work really well as there are numerous ways of representing complex structures in a simpler fashion. There is even an area where focus+context based solutions can provide helpful insight in the huge amount of data often found in flows.

Acknowledgements

I would like to take the opportunity to thank some people that helped me not only understand the complexities within this thesis but also helped me define and find solutions to problems I faced. Thanks to my fellow students for providing healthy discussion and a good reason to keep working.
I would really like to thank professor Hellwig Hauser for guiding me towards a final goal and providing me feedback and discussion on how to organize and what to focus on when formulating such a big project.
I would also like to express a special thanks toward my supervisor, Andrea Brambilla for finding solutions to problems I had and helping me not only figure out the basics but the complex answers often needed to complete this thesis. I would also give a special thanks for providing me with interesting information and papers that kept me interested in this project, Thank you.

Bibliography

[1] Ralph Abraham and Christopher D Shaw. *Dynamics: The Geometry of Behavior*. Number 1. Addison-Wesley, 1992.

[2] Andrea Brambilla, Robert Carnecky, Ronald Peikert, Ivan Viola, and Helwig Hauser. Illustrative flow visualization : State of the art , trends and challenges. *Proceedings of Eurographics State of the Arts Reports*, 2012.

[3] D Eberly, R Gardner, B Morse, S Pizer, and C Scharlach. 1 the need for ridges in image analysis. *Analysis*, 4(4):1–24, 1994.

[4] Cinzia G Farnetani and Henri Samuel. Lagrangian structures and stirring in the earth's mantle. *Earth and Planetary Science Letters*, 206(3-4):335–348, 2003.

[5] A Fuhrmann and E Groller. Real-time techniques for 3d flow visualization. *Proceedings Visualization 98 Cat No98CB36276*, pages 305–312, 1998.

[6] Jacob D Furst and Stephen M Pizer. *Marching Ridges*, pages 22–26. 2001.

[7] Christoph Garth, Han Krishnan, Xavier Tricoche, Tom Bobach, and Kenneth I Joy. Generation of accurate integral surfaces in time-dependent vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1404–1411, 2008.

[8] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. *A Non-photorealistic Lighting Model For Automatic Technical Graphics*. 1998.

[9] G Gorla, V Interrante, and G Sapiro. Texture synthesis for 3d shape representation, 2003.

[10] M A Green, C W Rowley, and G Haller. Detection of lagrangian coherent structures in three-dimensional turbulence. *Journal of Fluid Mechanics*, 572(-1):111, 2007.

[11] Melissa A Green, Clarence W Rowley, and Alexander J Smits. Using hyperbolic lagrangian coherent structures to investigate vortices in bioinspired fluid flows. *Chaos Woodbury Ny*, 20(1):017510, 2010.

[12] G Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248–277, 2001.

[13] G Haller and G Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3-4):352–370, 2000.

[14] H Hauser and M Mlejnek. *Interactive Volume Visualization of Complex Flow Semantics*, pages 191–198. 2003.

[15] J P M Hultquist. Constructing stream surfaces in steady 3d vector fields. *Proceedings of IEEE Visualization 1992*, pages 171–178, 1992.

[16] Mathias Hummel, Christoph Garth, Bernd Hamann, Hans Hagen, and Kenneth I Joy. Iris: illustrative rendering for integral surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1319–1328, 2010.

[17] C Jones and Kwan-Liu Ma Kwan-Liu Ma. Visualizing flow trajectories using locality-based rendering and warped curve plots. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1587–1594, 2010.

[18] Stuart Kent. Lagrangian coherent structures : Generalizing stable and unstable manifolds to non-autonomous dynamical systems dynamical systems : Background. *Spring*, pages 1–15, 2008.

[19] Gordon Kindlmann, Xavier Tricoche, and Carl-Fredrik Westin. Anisotropy creases delineate white matter structure in diffusion tensor mri. *Medical Image Computing and Computer-Assisted Intervention*, 9(Pt 1):126–133, 2006.

[20] R M Kirby, H Marmanis, and David H Laidlaw. *Visualizing Multivalued Data from 2D Incompressible Flows Using Concepts from Painting*, pages 333–340. IEEE Computer Society Press, 1999.

[21] J Kruger, P Kipfer, P Konclratieva, and R Westermann. A particle system for interactive visualization of 3d flows. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):744–756, 2005.

[22] A. Kuhn, C. Rössl, T. Weinkauf, and H. Theisel. A benchmark for evaluating FTLE computations. In *Proc. IEEE Pacific Visualization*, pages 121–128, Songdo, Korea, February 2012.

[23] Robert S Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H Post, and Daniel Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[24] Doug Lipinski and Kamran Mohseni. A ridge tracking algorithm and error estimate for efficient computation of lagrangian coherent structures. *Chaos Woodbury Ny*, 20(1):017504, 2010.

[25] Peter Majer. *A Statistical Approach to Feature Detection and Scale Selection in Images*. PhD thesis, der Universität Göttingen, 2000.

[26] Oliver Mattausch, Thomas Theußl, Helwig Hauser, and Eduard Gröller. Strategies for interactive exploration of 3d flow using evenly-spaced illuminated streamlines. *Spring Conference on Computer Graphics*, pages 213–222, 2003.

[27] Tony McLoughlin, Robert S. Laramee, Ronald Peikert, Frits H. Post, and Min Chen. Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[28] Chris Muelder and Kwan-Liu Ma. Interactive feature extraction and tracking by utilizing region coherency. *2009 IEEE Pacific Visualization Symposium*, pages 17–24, 2009.

[29] R Peikert and M Roth. *The "Parallel Vectors" operator-a vector field visualization primitive*, volume 99, pages 263–270. Ieee, 1999.

[30] Ronald Peikert and Filip Sadlo. *Height Ridge Computation and Filtering for Visualization*, volume eds, pages 119–126. IEEE, 2008.

[31] Frits H Post, Benjamin Vrolijk, Helwig Hauser, Robert S Laramee, and Helmut Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[32] C Rezk-Salama, P Hastreiter, C Teitzel, and T Ertl. Interactive exploration of volume line integral convolution based on 3d-texture mapping. *Proceedings Visualization 99 Cat No99CB37067*, Li(section 6):233–240, 1999.

[33] P Rheingans and D Ebert. Volume illustration: nonphotorealistic rendering of volume models, 2001.

[34] Filip Sadlo and Ronald Peikert. Efficient visualization of lagrangian coherent structures by filtered amr ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1456–1463, 2007.

[35] Jan Sahner, Tino Weinkauf, Nathalie Teuber, and Hans-Christian Hege. Vortex and strain skeletons in eulerian and lagrangian frames. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):980–990, 2007.

[36] Tobias Salzbrunn, Heike Janicke, Thomas Wischgoll, and Gerik Scheuermann. *The State of the Art in Flow Visualization: Partition-Based Techniques*, pages 75–92. SCS Publishing House e.V., 2008.

[37] A Sanna, B Montrucchio, and P Montuschi. A survey on visualization of vector elds by texture-based methods. *Citeseer*, 2000.

[38] Benjamin Schindler, Ronald Peikert, Raphael Fuchs, and Holger Theisel. Ridge concepts for the visualization of lagrangian coherent structures. In

Topological Methods in Data Analysis and Visualization II. Springer Berlin Heidelberg, 2012.

[39] Marc Schirski, Torsten Kuhlen, Martin Hopp, Philipp Adomeit, Stefan Pischinger, and Christian Bischof. Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets. *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry VRCAI 04*, 1(212):141, 2004.

[40] W J Schroeder, C R Volpe, and W E Lorensen. The stream polygon-a technique for 3d vector field visualization, 1991.

[41] S.C. Shadden. Lagrangian coherent structures: Analysis of time-dependent dynamical systems using ftle. http://mmae.iit.edu/shadden/LCS-tutorial/, 2005.

[42] Francois Lekien Shawn C. Shadden and Jerrold E. Marsden. Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(4):271–304, 2005.

[43] Mario Costa Sousa and John W Buchanan. Computer-generated graphite pencil rendering of 3d polygonal models. *Scientist*, 18(3):195–207, 1999.

[44] Nikolai A Svakhine, Yun Jang, David S Ebert, and Kelly P Gaither. Illustration and photography inspired visualization of flows and volumes. *Methodology*, pages 687–694, 2005.

[45] Shyh-Kuang Ueng, Christopher Sikorski, and Kwan-Liu Ma. Efficient streamline, streamribbon, and streamtube constructions on unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):100–110, 1996.

[46] Timothy Urness, Victoria Interrante, Ivan Marusic, Ellen Longmire, and Bharathram Ganapathisubramani. *Effectively visualizing multi-valued flow data using color and texture*, pages 115–121. IEEE Computer Society, 2003.

[47] Jishang Wei. A sketch-based interface for classifying and visualizing vector fields. *Strategy*, pages 129–136, 2010.

[48] D Xue, C Zhang, and R Crawfis. Rendering implicit flow volumes. *Proc IEEE Visualization 04 Conference*, pages 99–106, 2004.

[49] Zhonglin J Zhang and John Dabiri. Identification of lagrangian coherent structures around swimming jellyfish from experimental time-series data. *Mechancial Engineering*, 2008.

List of Figures

List of Tables