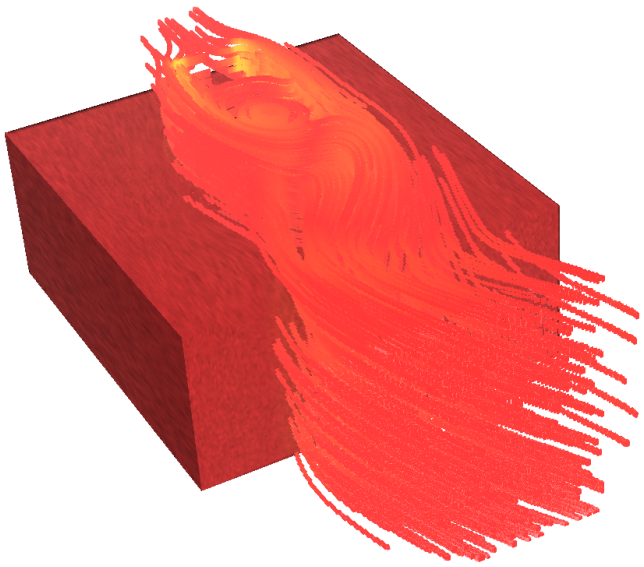
AN ABSTRACTION SPACE FOR FLUID FLOW VISUALIZATION

MATTHEW VAN DER ZWAN



Master Thesis

August 2011

Supervisors:
Dr. T. Isenberg
Prof. dr. A.E.P. Veldman
Dr. H. Bekker

ABSTRACT

The increase in computational power available in computers over the past decades allows researchers to run simulations with an increasing amount of grid points. While this improves the accuracy of the simulation results, it poses a challenge when visualizing these results. Traditional flow visualization techniques such as Line Integral Convolution (LIC) require a lot of time to generate the corresponding representation of the flow. Furthermore, visualizing these representations is also demanding. Therefore, other representations of fluid flow have been proposed, such as streamlines and flow topology representations. These methods reduce the amount of data which is visualized and thereby increase performance, but the resulting visualization might be more difficult to understand.

In this thesis we describe an abstraction space for fluid flow visualization. The goal of this abstraction space is to allow continuous transitions between different representations of the flow, where each representation corresponds to a level of structural abstraction. This abstraction space enables users to understand the relation between the different representations. Besides structural abstraction, we also employ techniques which enhance spatial perception for the different representations. We discuss how to enable a continuous transition between different fluid flow representations and how this structural transition can be combined with the techniques to enhance spatial perception. Throughout this thesis, we present results created with our realization of the abstraction space.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

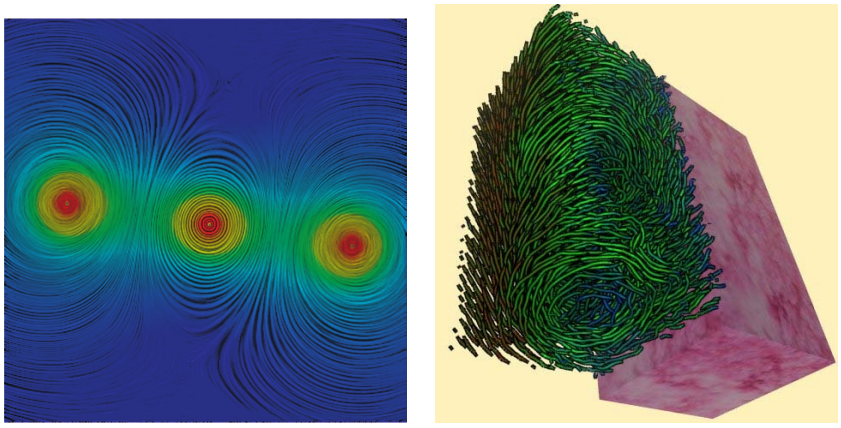
For a long time, researchers have been interested in studying the behaviour of fluids around objects. For example, the flow of air around an airfoil or water in a water way are cases which need to be understood in order to create the most efficient or sturdy solution. The oldest way to study these phenomena is by doing experiments. For the previous examples this can be done in a wind tunnel or using scale models, as shown in Figure 1a and Figure 1b respectively. However, these methods require a lot of time and are not always accurate since some physical phenomena will not appear at the size of the model.

(a) (b)

Figure 1: Two examples of flow experiments: (a) airflow past an airfoil, visualized by releasing smoke, (b) water through a water way.

An alternative approach to gain insight into flow behaviour is by using the Navier-Stokes equations which describe the motion of fluids [Batchelor, 1967]. However, Navier-Stokes equations can only be solved by hand when they are simplified. Therefore, people try to solve these equations using computers because this approach is faster than performing experiments and more accurate than both experiments and manually solving the simplified Navier -Stokes equations.

Solving the Navier-Stokes equations can be done using several techniques from numerical mathematics, such as finite differences [Chorin, 1968] or finite elements [Girault and Raviart, 1986]. For all techniques, an increase in accuracy comes with an increase in computational complexity. To obtain sufficient resolution of the flow details, the number of grid points on which the simulation is performed needs to be increased. The increasing computational power available in modern computers allows researchers to run simulations with an increasing amount of grid points, allowing a better insight in, for instance, turbulent flow.



(a) (b)

Figure 2: Two examples of commonly employed flow visualization techniques: (a) two-dimensional Accelerated Unsteady Flow Line Integral Convolution, an improvement upon LIC, from Liu and Moorhead [2005], (b) three-dimensional streamlines showing the flow in the vicinity of a block, from Mattausch et al. [2003].

As a means to study the results of the flow simulations, often images or movies are created to depict phenomena in the flow that are of interest to the user. For example, *Line Integral Convolution* (LIC) by Cabral and Leedom [1993] can be used to visualize the flow pattern along a plane as shown in Figure 2a. This image is created using an improved version of the technique by Liu and Moorhead [2005] where a noise texture is smeared out according to the direction of the flow, providing a general overview of the flow behaviour. This technique has also been applied to three-dimensional datasets, for example, by Helgeland and Andreassen [2004].

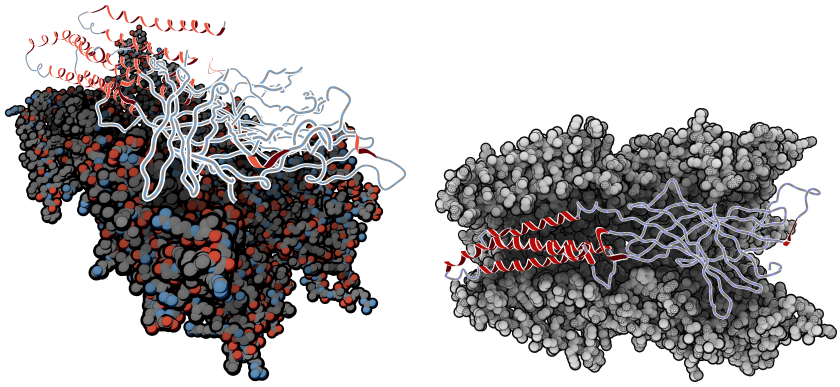
Other techniques that are traditionally used to visualize flow behaviour are stream lines [Jobard and Lefer, 1997] and stream surfaces [Hultquist, 1992]. For these techniques, a seed point or a more complex object in the case of flow surfaces is inserted into the flow and moved along the flow. This results in lines or surfaces which show the pattern of the flow as can be seen in Figure 2b. For small simulations it is also possible to study flow behaviour by showing the flow's vector field.

While the increasing resolution of the flow simulations provides a higher accuracy of the results, the visualization of the data becomes more complex. For instance, studying the vector field becomes difficult due to occlusion effects and the vectors on the outer regions occlude view of the vectors in the inner part of the data. Another factor that increases when the resolution of the simulation data increases is the computation time for techniques such as LIC, especially for three-dimensional data sets. Several techniques have been proposed to improve LIC, reducing both computational time and occlusion issues

[Helgeland and Andreassen, 2004; Liu and Moorhead, 2005; Stalling and Hege, 1995].

Another approach to visualizing large data sets is applying abstraction, for instance, by only displaying regions that meet a certain criterion and tracking these over time [Post et al., 2003]. The amount of data that is displayed can also be reduced by determining the topological skeleton of the data set and visualizing this [Helman and Hesselink, 1989, 1991]. Abstraction using flow topology has also been applied to three-dimensional simulation results by Theisel et al. [2003], who also introduce the concept of saddle connectors which are streamlines that connect topologically interesting points in the flow.

Through abstraction, the amount of data to be displayed can be reduced, reducing occlusion issues and increasing the speed of the visualization [van der Zwan et al., 2011]. However, when solely using the abstracted form of the data, the context is lost and the abstracted form itself might be difficult to understand to inexperienced users. This is not only the case when applying abstraction to results of a fluid flow simulation but also for molecular data.



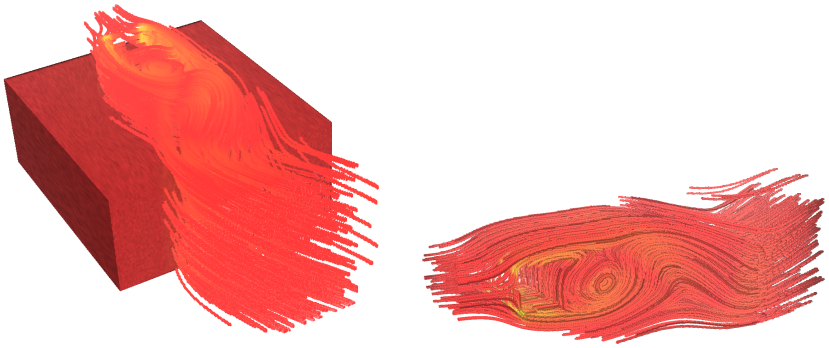
(a) (b)

Figure 3: Two examples using the molecular visualization abstraction spaces, both from van der Zwan et al. [2011]: (a) combination of two levels of structural abstraction, (b) focus and context by combining different levels of structural abstraction with different visual styles.

Previously, we have developed a technique for the interactive control of abstraction for protein data [van der Zwan et al., 2011]. We defined an *abstraction space* which allows continuous transitions through several stages of abstraction, going from a completely un-abstracted representation to a fully abstracted form. Besides providing continuous structural abstraction, the abstraction space for molecular visualization also allows users to change the visual style of the resulting image on a continuous scale. The visual style is controlled through two axes, one which adds depth cueing techniques and a second axis which controls the level of "illustrativeness". The level of "illustrativeness" ranges from photorealistic to black-and-white images. Applying the levels of abstraction locally allows users to create images where focus

is applied to the regions that are of interest, while retaining the rest of the molecule as context, as shown in Figure 3.

Inspired by the approach used to deal with abstraction for molecular data, we developed an abstraction space for fluid flow data. Similar to the abstraction space for molecular visualization, the abstraction space for fluid flow visualization provides interactive control of the level of abstraction applied. Besides allowing the user to select a global level of structural abstraction, we also allow to use a less abstracted visualization as context as shown in Figure 4a. We also provide additional techniques to enhance the perception of spatial relations in the resulting image as demonstrated in Figure 4b.



(a) (b)

Figure 4: Two examples using the fluid flow visualization abstraction spaces: (a) behaviour in a region of interest with context, (b) use of spatial perception enhancement techniques to show flow behaviour in a region of interest.

The remainder of this thesis is structured as follows: First, we discuss related work in Chapter 2. We start by giving an overview of important concepts in flow simulation, followed by an overview of abstraction in illustrative visualization and illustrative rendering techniques which are relevant for flow visualization. These flow visualization techniques are the topic of the last section in this chapter.

Chapter 3 describes the abstraction space for fluid visualization. First, we discuss the benefits of different flow representations. Then, we discuss how these representations can be combined to create a continuous transition and how this can be combined with technique to enhance spatial perception. Finally, we discuss two extensions to the abstraction space.

We discuss our realization of the abstraction space in Chapter 4 starting with the realization of the different flow representations. For each representation we discuss how to apply the appropriate techniques to enhance spatial perception. We conclude this chapter by describing how we realized the transitions between the different representations and the integration of the extensions presented in the previous chapter.

Visual results are presented in Chapter 5 as well as performance measurements and informal feedback. Finally, we discuss the results and suggest future work in Chapter 6.

2

RELATED WORK

This chapter gives an overview of work relevant for our research. First, we give a summary of some methods from computational fluid dynamics and flow properties that might help in understanding our research, in Section 2.1. After that, we discuss relevant illustrative rendering techniques and the application of abstraction in illustrative rendering in Section 2.2. Finally, we present an overview of flow visualization techniques in Section 2.3.

2.1 FLOW SIMULATION

The Navier-Stokes equations that describe the motion of fluids can only be solved analytically in special, simplified cases as mentioned before. Therefore, computers are used to solve the Navier-Stokes equations. We give a short description of two methods which can be used to solve the Navier-Stokes equations, finite differences and finite elements. These two techniques are somewhat different from each other but have in common that the running time of a simulation increases when we want to increase the accuracy of the solution.

When applying finite differences to solve a partial differential equation such as the Navier-Stokes equation, the partial derivatives in the equation are approximated [Chorin, 1968]. There are many ways in which theses derivatives can be approximated, with every method having a different accuracy which increases with the complexity of the approximation in most cases. The accuracy can also be influenced by the number of grid cells used for the simulation. Using more grid cells in the same domain results in smaller grid cells and a higher accuracy of the approximations and, therefore, provides a higher accuracy of the solution of the equation. However, increasing the number of grid cells also increases the time required to find a solution. The simulation results we use in our reference implementation are all calculated in this way.

Using finite element to solve a partial differential equation requires the domain to be divided into small parts called the finite elements on which the equation is solved [Girault and Raviart, 1986]. These finite elements can be, for example, triangles or quads in the two-dimensional case or tetrahedra or cubes when trying to find a three-dimensional solution. In contrast to using finite differences, the equation is first rewritten into a problem where a function needs to be found that solve an integral formulation called the *weak formulation*. Using a decomposition of the solution into basis functions, the integral can be

solved for every finite element after which the local solutions can be combined to form the global solution of the equation. The accuracy of this solution is influenced by the type of the used basis functions and increases when more complex basis functions are used, but using elements of a different size also influences the accuracy. Again, the time required to find a solution increases with the accuracy of this solution.

2.2 ILLUSTRATIVE RENDERING AND ABSTRACTION

Studying the results of a flow simulation directly is difficult for small domains and impossible for large domains, since it is too hard to see all the effects by studying the raw data. Therefore, the results are often visualized using flow visualization techniques as discussed in Section 2.3. In this section we discuss some more general visualization techniques which are also used in flow visualization.

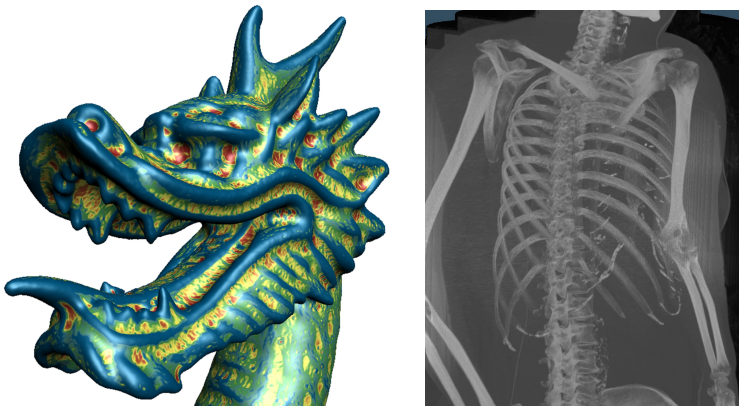
Illustrative rendering is a subfield of *non-photorealistic rendering* (NPR) which takes inspiration from traditional illustration techniques [Rautek et al., 2008]. Gooch and Gooch [2001] and Strothotte and Schlechtweg [2002], for example, give an overview of NPR techniques.

Illustrative rendering covers a wide field of techniques, which can be focused on creating artistic images [Nienhaus and Döllner, 2004] or aid the understanding of scientific data such as geometric data [Patel et al., 2008] or protein data [Weber, 2009; van der Zwan et al., 2011]. In scientific visualization, illustrative rendering is often used to enhance the visualization such as providing the users with depth cues or indicating regions of interest.

2.2.1 *Volume visualization*

While volume visualization in itself is not an illustrative rendering technique, it is often combined with these techniques to create images which show only the important parts of the volume and, therefore, are not photo-realistic. Traditionally, volume visualization (or volume rendering) is used most often in medical applications to visualize the data acquired using, for example, MRI scanners [Zhang et al., 2011]. The goal of volume visualization is to create a two-dimensional image showing the three-dimensional input volume, or parts of this volume. For example, Rezk-Salama et al. [1999] allow the investigation of three-dimensional fluid volumes through the use of clip planes. Hadwiger et al. [2005] introduced a hardware-accelerated technique to visualize regions with the same value, called *isosurfaces*, in the dataset as shown in Figure 5a.

The volume rendering techniques that are relevant to our work are the direct volume rendering techniques. These techniques create a two-dimensional image from three-dimensional data without computing



(a) (b)

Figure 5: Two examples of volume visualization: (a) Isosurface, from Hadwiger et al. [2005]. (b) Maximum Intensity Projection, from Parker et al. [1999].

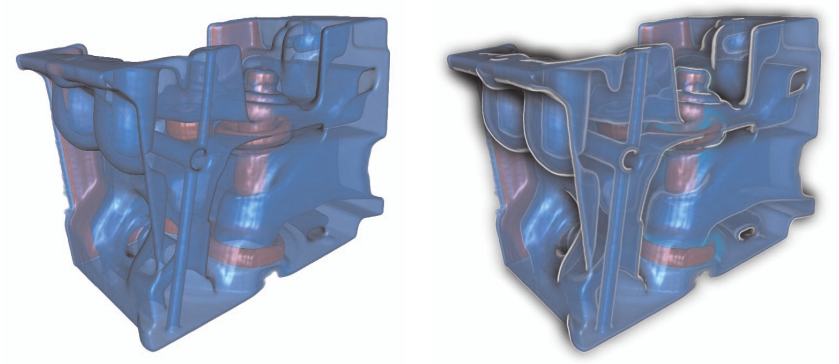
any intermediate representations [Kaufman and Mueller, 2005], as opposed to, for instance, surface rendering techniques.

The first volume rendering techniques using graphics hardware were introduced by Cullip and Neumann [1993] and Cabral et al. [1994]. Most modern direct volume renderers are based on *ray-casting* introduced by Levoy [1988]. With ray-casting, a ray is computed through the image plane for every pixel on this image plane. If this ray hits the volume the resulting value of the pixel on the image plane will be a function of the values of the volume along this ray. This *ray function* can be the maximum value encountered along the ray, resulting in the *maximum intensity projection* Wallis et al. [1989]. However, the ray function can also simulate the behaviour of a real light ray travelling through the volume, in which case it is called a *compositing function*.

Regardless of the choice of ray function we also need a way to map the values of the volume data (or *intensity*) to color and opacity in the resulting image. The function which does this is called the *transfer function*. The final color of the image pixel is determined by a combination of the ray function and the transfer function. For maximum intensity projection, the transfer function is applied to the maximum intensity found by the ray function, while for the compositing ray function the values of the transfer function corresponding to intensities along the ray are blended to create the final pixel color.

While the combination of a suitable transfer function and ray function can give a good view of the relevant data inside the volume, the resulting images do not show the shape of the three-dimensional objects in the volume. Therefore, global shading models are commonly applied. Most of these global shading models work by approximating the local gradient and use this in a traditional shading model such as

Phong shading [Phong, 1975]. Figure 5a and Figure 6a show examples of shaded volume rendering. For the current research, we do not use a global shading model because they are computationally expensive and we can apply some of the computationally less-expensive methods presented next.



(a) (b)

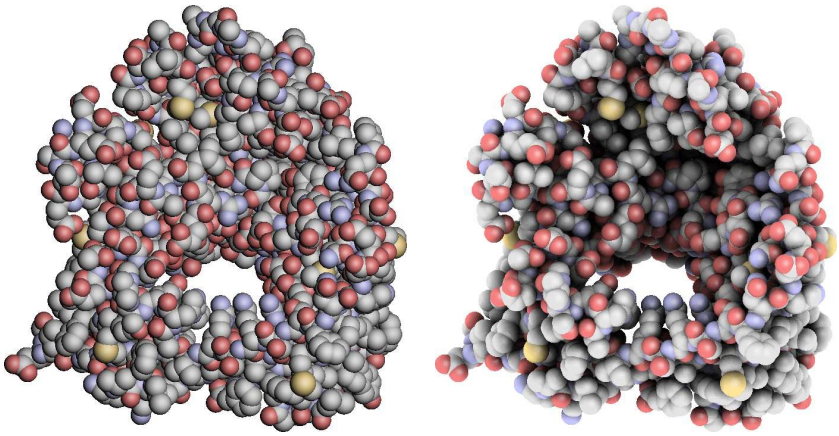
Figure 6: A comparison of shaded volume visualization with and without halos, from Bruckner and Gröller [2007]: (a) Without halos, (b) With halos.

2.2.2 *Enhancement of spatial perception*

For the volume rendering techniques discussed in the previous section, the use of global shading models might help convey the shape of objects, but spatial relations are not always clear. Bruckner and Gröller [2007] propose the use of *flexible volumetric halos* to enhance perception of spatial relations in volume rendered images. These volumetric halos are inspired by halos used in traditional illustration such as medical illustration but can also be applied to technical images as shown in Figure 6b. The halos in this example emphasize the shape and depth relations of individual parts because the halos block the objects further away from the viewer.

Halos are not only used for enhancing depth perception in volume visualization. Since their introduction to computer graphics by Appel et al. [1979], halos have been applied in different application domains. For example, halos are used to enhance perception of spatial relations for line drawings [Elber, 1995; Everts et al., 2009] or the spatial relations of atoms in molecular visualization [Tarini et al., 2006; van der Zwan et al., 2011].

Besides using halos, Tarini et al. [2006] and van der Zwan et al. [2011] use shadows to enhance spatial perception. These shadows are calculated using *ambient occlusion*, a technique used to approximate the amount of ambient light reaching a point on the object [Kajiya, 1986]. Ambient occlusion is calculated by determining how much light



(a) (b)

Figure 7: A comparison of molecular visualization with (a) and without (b) ambient occlusion, from Tarini et al. [2006].

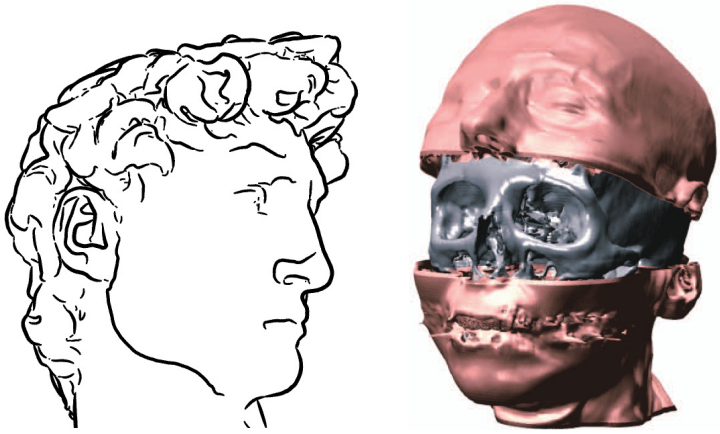
from the environment (ambient light) can reach each point on the surface. This results in dark regions which are hard to reach for the light and lighter regions, enhancing spatial perception as shown in Figure 7. Although ambient occlusion is computationally expansive to compute, it can be precomputed for static geometry. Since the continuous transition between representations results in changing geometry, we would have to compute the ambient occlusion again for each level of structural abstraction so we chose not to use ambient occlusion.

2.2.3 *Abstraction*

In their paper about illustrative visualization, Rautek et al. [2008] distinguish two forms of abstraction: low-level and high-level abstractions. *Low-level abstractions* focus on how to render features of interest, without changing geometry. For example, the suggestive contours introduced by DeCarlo et al. [2003] which convey the shape of an object by highlighting contours as shown in Figure 8a.

High-level abstractions deal with what to render [Rautek et al., 2008], some parts of the data might be removed from the visualization to reveal other, more important parts. In contrast to low-level abstractions, high-level abstractions might visualize parts of the data in different positions then they are originally. For example, the volume splitting technique described by Islam et al. [2007] is able to move certain parts of volume data to reveal important information which would otherwise be occluded.

Van der Zwan et al. [2011] combine both low-level and high-level abstraction in their *abstraction space* for molecular visualization. This abstraction space has three axes which control how the molecule is



(a) (b)

Figure 8: Two forms of abstraction: (a) low-level abstraction: suggestive contours, from DeCarlo et al. [2003], (b) high-level abstraction: volume splitting, from Islam et al. [2007].

visualized. The first axis controls the level of structural abstraction and determines which atoms and bonds in the molecule are shown and what they look like being an example of high-level abstraction. The second axis controls the level of enhancement of structural perception: dependent on this level, ambient occlusion and halos are added, providing depth cues to the user. Since the second axis deals mostly with how things are drawn it is an example of low-level abstraction, just as the third axis which allows to choose a drawing style, ranging from photo-realistic to black-and-white images. This abstraction space for molecular visualization was the inspiration for the fluid flow abstraction space created during this research.

2.3 FLOW VISUALIZATION

An important distinction in flow visualization methods is the time-dependence of the input data. Methods that use time-independent data are in general able to achieve a higher frame rate due to the fact that less data transfer is needed, whereas time-dependent data needs to be updated (at least partially) for each time step. Laramee et al. [2004] refer to techniques using huge time-dependent (also referred to as unsteady) data as one of the areas in flow visualization that need additional work.

In their paper describing the state of the art in flow visualization, Laramee et al. [2004] divide the field of flow visualization into four categories, depending on the different needs of the users:

- *Direct flow visualization*: The visualization is based directly on the raw flow data.

- *Dense, texture-based flow visualization*: A texture is computed from the flow data, which is then used to create a dense representation of the flow, similar to direct flow visualization.
- *Geometric flow visualization*: Geometric objects, for example, lines or surfaces, are used to represent properties of the flow.
- *Feature-based flow visualization*: Special features are extracted from the flow data and used to visualize the flow.

Figure 9 gives a graphical overview of this classification. In the following sections, we will discuss each of these categories in detail as well as give examples for all categories.

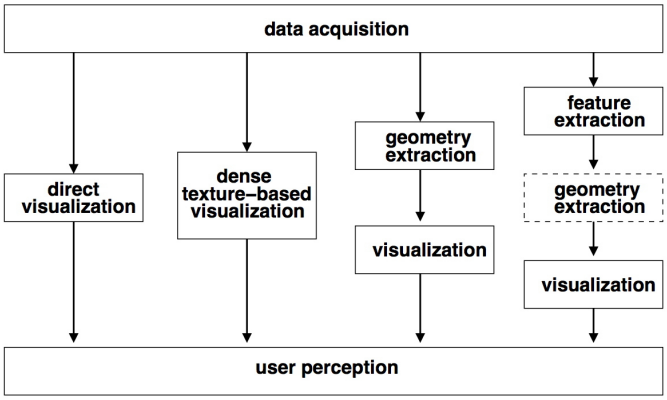
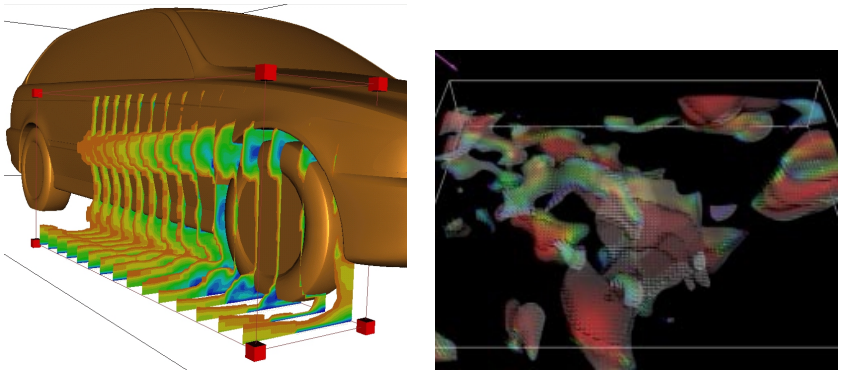


Figure 9: Classification of flow visualization techniques and their relation to the input data, from Laramee et al. [2004].

2.3.1 *Direct flow visualization*

Flow visualization techniques that make direct use of the flow data are referred to as direct flow visualization techniques. For these techniques, no preprocessing is performed on the data before visualization, where techniques from the other categories all require some form of preprocessing.

According to Hauser et al. [2003], an intuitive and commonly used direct flow visualization technique is *color coding*. Using color coding, important flow properties such as flow direction, flow magnitude, or pressure are mapped to colors. For two-dimensional data, this results in an easy to understand visualization of the data. However, when displaying all available data for three-dimensional data, the information on the outside of the flow domain will occlude the inner part of the domain. Schulz et al. [1999] solve this problem by allowing the user to place a measurement probe in the region of interest. This measurement probe can either be a slice or a cube, resulting in a two-dimensional or three-dimensional color coded representation of the local flow, respectively, as shown in Figure 10a.



(a) (b)

Figure 10: Two examples of direct flow visualization: (a) Flow of air past the wheel housing of a car. Volume rendering on slices defined by a cubical measurement probe, with colors indicating the flow velocity, from Schulz et al. [1999]. (b) Glyph visualization of a weather dataset. The arrow in the upper-left corner indicates the studied direction, from Boring and Pang [1996].

Another method to visualize vector fields is displaying the vectors using *glyphs* such as arrows or spheres [Hauser et al., 2003] where properties of the flow can be mapped to properties of the glyphs. For example, when using arrows, these can point in the direction of the flow and their length can be proportional to the magnitude of the flow at that point. This technique can also be combined with color coding of the glyphs to visualize more flow properties such as pressure or vorticity. Similar to the color coding method, glyphs also suffer from occlusion issues. This problem has been addressed, for example, by Boring and Pang [1996] by highlighting vectors based on directional information. The user selects a direction and vectors that point in a different direction will appear dimmer, placing the focus on vectors that point in the selected direction as shown in Figure 10b. In this research we do not use direct flow visualization, although we use color to show properties of the flow in combination with other visualization techniques. The vector highlighting technique by Boring and Pang [1996] was the inspiration to use fog as depth-cueing technique.

2.3.2 *Dense, texture-based flow visualization*

Dense, texture-based flow visualization techniques derive a texture from the flow data, which is used to visualize flow behaviour. One of the first dense, texture-based flow visualization techniques is *spot noise*, introduced by van Wijk [1991]. The name of the technique comes from the spot primitives which are at its core and the fact that these spots are randomly distributed over the domain. Similar to the glyphs mentioned before, properties of the spots can be derived from properties of the flow. Depending on the choice of spot and the

mapping of flow properties to spot properties, different aspects of the flow can be studied on a global level as shown in Figure 11.

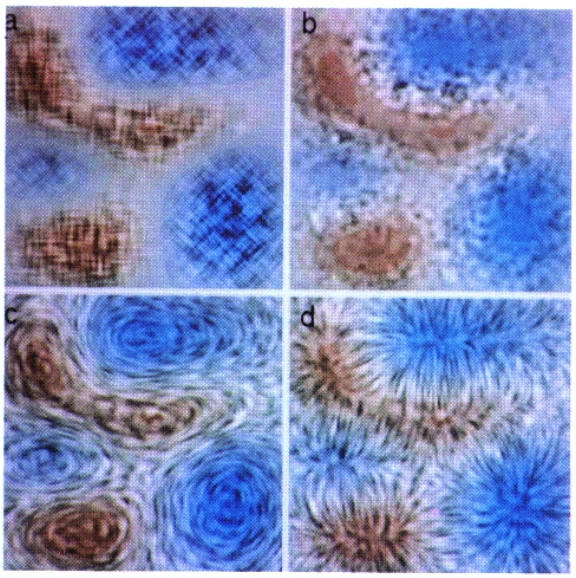
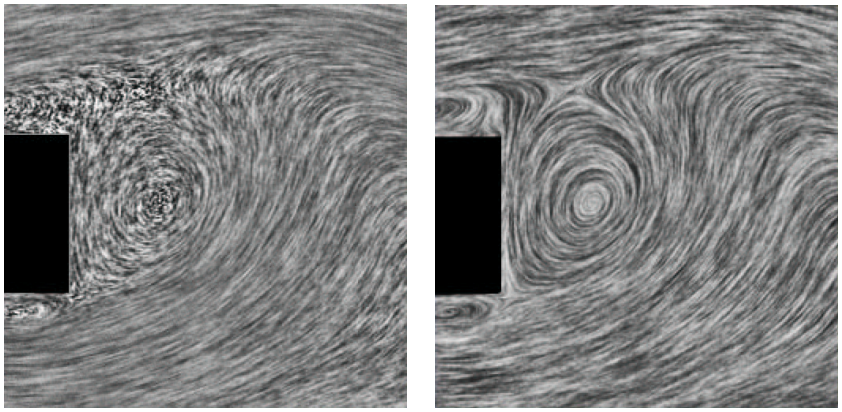


Figure 11: Spot noise, an example of dense, texture-based flow visualization, from van Wijk [1991]. The colors show the value of a scalar field, while the spot noise depicts attributes or interpretations of the data: (a) Spot type indicates sign, (b) Variance of texture scaled by norm of the gradient, (c) Spots stretched according to flow, (d) Spots stretched according to velocity potential.

Images similar to Figure 11c can be created using a technique developed by Cabral and Leedom [1993] called *line integral convolution* (LIC). The LIC method uses a vector field and a white noise texture as input. For every pixel, a local forward and backward streamline is derived from the vector field. The intensity of a pixel is calculated by integrating over this streamline while using an appropriate kernel. Several enhancements have been introduced to this basic LIC algorithm. For example, Forssell and Cohen [1995] extend the method to curvilinear surfaces and time-dependent flow and use texture mapping hardware to run it in real time.

While the images created with spot noise and LIC might look similar at first glance, there are, however, differences between the results created with these two methods as shown in Figure 12. The image generated with spot noise provides better understanding of the flow magnitude because this directly affects the stretching of the spots, resulting in high contrast between images with slow and fast flow. The basic LIC technique, on the other hand, does not show magnitude since only the (normalized) direction of the flow is used during computation. However, LIC has the advantage that the direction of flow is also visible for regions with low velocity. Since we can use different cues to show the magnitude of the flow, for instance using color we choose to

use the LIC technique as it is harder to encode direction information in the spot noise representation.



(a) (b)

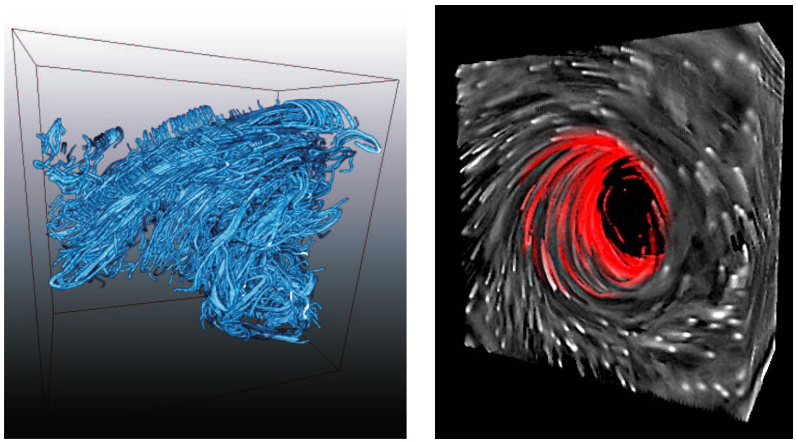
Figure 12: A visual comparison between (a): spot noise and (b): LIC, from Laramee et al. [2004].

One of the major disadvantages of the original LIC method is the computational time needed to create the resulting texture, especially when applied to three-dimensional data sets. Therefore, Stalling and Hege [1995] propose a different way of computing the LIC texture, which they call fast-LIC. They report their method to be an order of magnitude faster than previous methods [Stalling and Hege, 1995]. This reducing of the size of the visualized volume is a form of abstraction which is made necessary by the occlusion issues that occur for non-abstracted representations.

Applying LIC to three-dimensional data sets leads to occlusion issues when using a dense input texture. Interrante and Grosch [1998] propose to use a sparse input texture which only has points in the chosen *region of interest* (ROI). Using a sparse input texture results in an image where we can also look inside the LIC volume, instead of resulting in a solid LIC image where only the outer surface is visible. They add halos to the resulting image by performing LIC simultaneously over two input textures, using the second texture for halo computation. Helgeland and Andreassen [2004] also use sparse input textures but change the transfer function of the volume visualization to generate limb-darkening halos as shown in Figure 13a. Since changing the transfer function is less computationally expensive then computing a second LIC volume and some of the depth cueing techniques mentioned before such as ambient occlusion we use these limb-darkening halos for our technique.

Another dense, texture-based flow visualization technique is *Image Based Flow Visualization* (IBFV), introduced by van Wijk [2002]. This technique has been designed to make use of graphical hardware and works by warping the image of the previous frame according to the

flow field while blending in a new white noise image. Streamlines or particles can be simulated using dye injection, also for time-dependent data. While the original method uses two-dimensional data, IBFV has been extended to two-dimensional surfaces in three-dimensional data by van Wijk [2003] and to complete three-dimensional data sets by Telea and van Wijk [2003]. An example of three-dimensional IBFV can be seen in Figure 13b. We chose not to use IBFV for our abstraction space because other techniques such as seed LIC provide a better perception of the spatial relations when combined with depth-cueing techniques such as the limb-darkening halos.



(a) (b)

Figure 13: Two examples of three-dimensional dense, texture-based flow visualization techniques: (a) seedLIC, from Helgeland and Andreassen [2004], (b) 3-D IBFV, from Telea and van Wijk [2003].

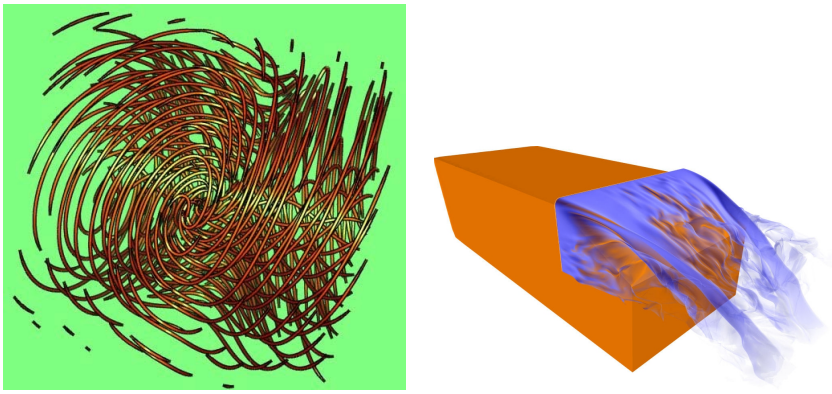
2.3.3 *Geometric flow visualization*

McLoughlin et al. [2010] give a comprehensive overview of geometric flow visualization techniques. As the name suggests, geometric methods compute geometric objects which represent flow behaviour, commonly as preprocessing step. These computation start from a set of seeding points from which streamlines are computed. The computed streamlines can then be visualized directly or used to create other geometric objects such as stream surfaces, for example.

The placing of seed points in the domain is very important since incorrectly placed seedpoints can lead to visual clutter and occlusion problems [McLoughlin et al., 2010]. Jobard and Lefer [1997] propose an image-based technique to solve the clutter problem for streamlines in two-dimensional images. Their method allows the user to select an appropriate maximum distance between streamlines which is used as a threshold for the insertion of new streamlines. The new streamline is only inserted if the distance to all its neighbours is above this threshold.

Jobard and Lefer [2001] also have extended their technique to support varying levels of line density, allowing the line density to be influenced by flow properties such as the velocity.

Mattausch et al. [2003] have expanded the streamline placement techniques by Jobard and Lefer [1997, 2001] to three-dimensional data sets. Their technique also allows the interactive selection of streamline density, either by the user or based on flow properties. They use the illuminated streamlines technique, introduced by Zöckler et al. [1996], to render the streamlines. This technique allows the generation of streamlines which are drawn as lines to be shaded as if they were tubes, aiding users in better perceiving depth relations between the streamlines. Mattausch et al. [2003] expand the illuminated streamlines technique with halos to further enhance depth perception, as shown in Figure 14a. We use streamlines as one of representation of the fluid flow in our abstraction space, but do not use the global illumination.



(a) (b)

Figure 14: Two examples of geometric flow visualization techniques: (a) Illuminated streamlines with halos, from Mattausch et al. [2003], (b) Smoke surface showing the flow on a block, from von Funck et al. [2008].

Another commonly used geometric primitive used to visualize flow is the stream surface [Hultquist, 1992]. Stream surfaces are computed by seeding a line into the flow and integrating this line according to the flow. This new front line is then connected to the previous front line, resulting in a surface showing flow behaviour after several integration steps. During the integration, the streamlines that together form the stream surface might converge or diverge. Hultquist [1992] proposes a technique for dynamic removal and insertion of particles to keep the density of triangles in the surfaces fairly constant. Von Funck et al. [2008] propose a technique for visualizing time dependent data called smoke surfaces which does not require point insertion and removing. They avoid these expansive operations by using the distance to other points to guide opacity instead of triggering an insertion or deletion of a new point. An image created using their technique is shown in Figure 14b. Integration of streamlines is used in the computation of

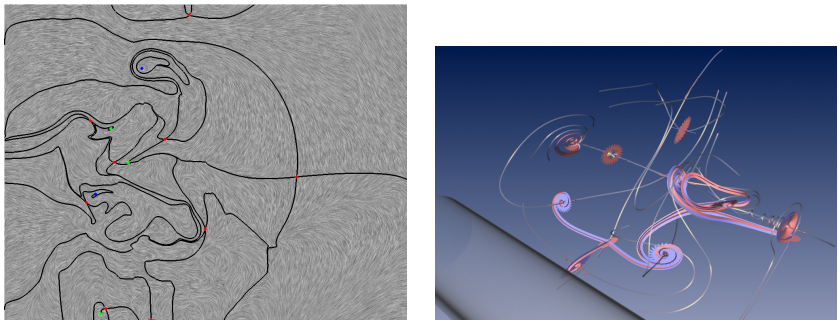
a representation used in our abstraction space and there we use the techniques for dynamic removal and insertion of particles proposed by Hultquist [1992].

2.3.4 *Feature-based flow visualization*

Similar to geometric flow visualization techniques, feature-based flow visualization techniques apply pre-processing to the data prior to visualization. In this pre-processing step, relevant *features* in the data are extracted. Features are (physical) phenomena or structures which are present in the data set [Post et al., 2003]. For example, these features can be critical points [Helman and Hesselink, 1989] or vortices [Banks and Singer, 1994] in the flow.

The features extracted from the flow data can be used as the basis for an abstracted visualization of this data, or enhance the visualization by making users aware of the detected features. An example of the latter is presented by Sadarjoen and Post [1999], whose technique is capable of detecting vortices and uses icons to indicate these vortices during visualization.

Helman and Hesselink [1989, 1991] introduced the idea of using vector field topology for fluid flow visualization. They presented a technique to extract critical points and proposed icons to visualize the different types of critical points in two-dimensional flows. Tricoche et al. [2001] propose a technique to simplify the topology by removing pairs of critical points according to a user-controlled relevance meausure. When combined with LIC to show the context as in Figure 15a, this provides a visualization of the topology without visual clutter and a better understanding of the flow then using only LIC. The goal of our research is to provide an intuitive way to combine these different representations and thereby enhance the user's understanding of the relation between the representations.



(a) (b)

Figure 15: Two examples of topology based flow visualization techniques: (a) Two-dimensional flow topology where colored spots indicate critical points and with LIC as context, from Tricoche et al. [2001], (b) Three-dimensional flow topology, with icons indicating critical points and saddle connectors, from Theisel et al. [2003].

Topology detection has also been applied to three-dimensional data sets, for example by Helman and Hesselink [1991] and Globus et al. [1991]. While detection of the topology in three-dimensional data is a straightforward generalization of the detection in two-dimensional data, visualization is more complicated; only showing the icons for the critical points does not show any context or relation between these points. On the other hand, visualizing the surfaces that separate the regions of uniform behaviour corresponding to the critical points (*seperation surfaces*) reintroduce visual clutters, even when applying transparency [Theisel et al., 2003].

Theisel et al. [2003] propose a technique which shows the connection between the critical points without suffering from visual clutter or occlusion issues. Instead of showing the full seperation surfaces, they only show the streamlines which are the intersections of these seperation surfaces. They call these streamlines *saddle connectors*. We use this technique because we think it can be very useful to researchers since it shows the most important features of the flow and their relation.

2.4 SUMMARY

In this chapter we gave an overview of different flow visualization techniques and tried to describe the upsides and downsides such as visual quality, depth perception and computation time of using the different techniques. We also described some general visualization techniques that are also applied in flow visualization. Combining the different flow visualization techniques presented in this chapter allows researchers to study the flow at different levels of abstraction with each level providing a different insight. Therefore, we describe how these techniques can be combined in the next chapter.

3 ABSTRACTION SPACE

The goal of this research is to develop an abstraction space for fluid flow visualization. This *abstraction space for flow visualization* has two axes, the first axis controls *structural abstraction* and the second one controls the level of *support of spatial perception*. The structural abstraction axis controls which flow visualization technique is used based on the position along the axis. We describe how we chose the structural representations and their order along the axis in Section 3.1. Support of spatial perception is the topic of Section 3.2. Extensions to the global abstraction space described in these sections are given in Section 3.3.

3.1 STRUCTURAL ABSTRACTION

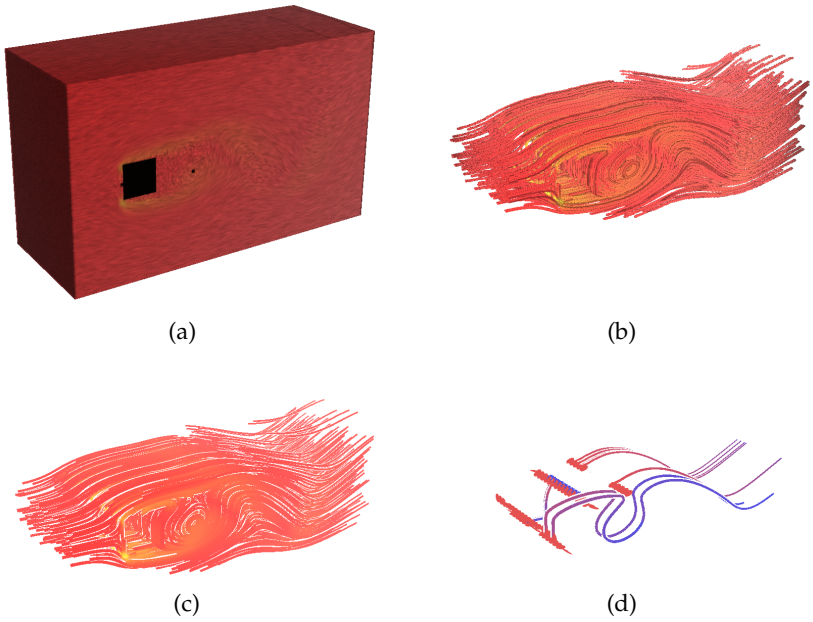
The goal of the structural abstraction axis is to enable the user to identify and study the important phenomena in the flow by allowing continuous transition through different representations of the flow. To provide information about the flow in every point of the domain there should be at least one representation which covers the entire domain. Therefore, one of the representation we use is the traditional LIC visualization of the entire domain (*dense LIC volume*) introduced by Cabral and Leedom [1993] as shown in Figure 16a. Combined with clip planes, this representation can be used to study local behaviour in detail [Rezk-Salama et al., 1999].

However, often researchers are only interested in the behaviour of the flow in specific regions such as, for example, regions with a lot of turbulence. Because we do not want the researchers to visually identifying such regions in a representation which shows the entire domain (such as the dense LIC volume), we wish to provide a representation that is capable of showing the behaviour in such a region of interest. Therefore, we include the sparse seedLIC visualization presented by Helgeland and Andreassen [2004] on our abstraction axis and allow users to specify a region of interest based on physical properties of the flow. Figure 16b shows an example of a region of interest visualized using seedLIC.

Although the seedLIC representation allows the user to focus on a region of interest, the resulting image still contains a lot of information in which the user should manually identify the important information. Therefore, we want to provide a representation of the flow that highlights the points in the flow where special behaviour occurs and the relations between these points, so the user can understand how the flow behaves around these points. To this end, we chose to use the

flow topology visualization with topology connectors introduced by Theisel et al. [2003] as shown in Figure 16d.

To enable transitions along the axis of structural abstraction, we need to determine in what order the representations mentioned above should be placed on the structural abstraction axis. Increasing the amount of structural abstraction should lead to more abstracted visualizations, allowing the user to study only the most important parts of the flow. Therefore, we order the representations according to increasing abstraction, i. e. first the dense LIC representation, then seedLIC and finally topology. To enable seamless transition from the volumetric seedLIC representation to the geometric topology representation, we add a fourth representation: streamlines (Figure 16c). The streamlines show the same region of interest as the seedLIC representation because we use the streamlines which were computed during the seedLIC computation, but they can be combined with different depth cueing techniques as detailed below. Furthermore, the transition from streamlines to topology is possible because both are geometric representations and the topology connectors are a special case of streamlines.



(a) (b) (c) (d)

Figure 16: Increasing levels of structural abstraction: (a) traditional (dense) LIC showing the cylinder in the flow as a black object, (b) seedLIC in region of interest, (c) streamlines in region of interest, (d) flow topology with saddle connectors.

We have introduced the different representations we want to use and discussed an order along the structural abstraction axis which allow intuitive selection of the required level of structural abstraction. Now we can describe how the transitions between the different representations can be made. The first transition is the transition from

the traditional LIC representation, which covers the entire domain, towards the sparse seedLIC representation. We can make this transition by increasing the transparency of the LIC volume as we move along the structural abstraction axis, thereby revealing the seedLIC volume inside. This is possible because both the dense LIC volume as the seedLIC volume are computed using the same basic principle but with different seed points. For the dense LIC volume, the entire domain is covered, in contrast to the seedLIC volume which covers only a small part of the domain. However, the method used to determine the intensity of a voxel is the same for both LIC representations, the only difference being the amount of voxels which are covered.

The next transition, from the seedLIC volume to the streamlines works identical to the first transition; the opacity of the seedLIC volume is decreased until we only see the streamlines inside the volume. As mentioned when introducing the streamline representation, these streamlines are used in the calculation of the seedLIC volume. Therefore, the streamlines are positioned in the centers of the seedLIC bundles. When the opacity of the seedLIC volume is reduced, the streamlines inside the volume become visible.

The transition from streamlines to flow topology is more involved, because we need to transition from a large collection of streamlines to a small set of topology connectors and topology icons, using different visual styles at both ends of the transition. Therefore, this transition is performed in two stages. During the first stage, streamlines are removed until only the streamlines that are part of the flow topology, as saddle connectors, remain. Figures 17a-c provide a visual example of this transition. The removal of streamlines is based on a measure of importance. We chose the importance function to be the distance to the nearest saddle connector, because we found out during testing that this gives a transition which we experience as fluent and pleasant. When the level of structural abstraction increases, an importance threshold is calculated and all streamlines with an importance less than this threshold are omitted.This importance function first removes streamlines which are furthest away from the saddle connectors, removing the closer streamlines as well when structural abstraction increases. Other importance function are possible as well, for example, we might change our distance-based importance function to do precisely the opposite. Then, first the streamlines close to the saddle connectors would be removed, placing focus on these connectors, while using the streamlines further away as context. While this might not lead to the continuous transition we want to create, this allows researchers to study the relation between around critical points and behavior further away.

During the second stage of the transition from streamlines to flow topology we change the appearance of the saddle connectors while simultaneously introducing the icons which identify the critical points.

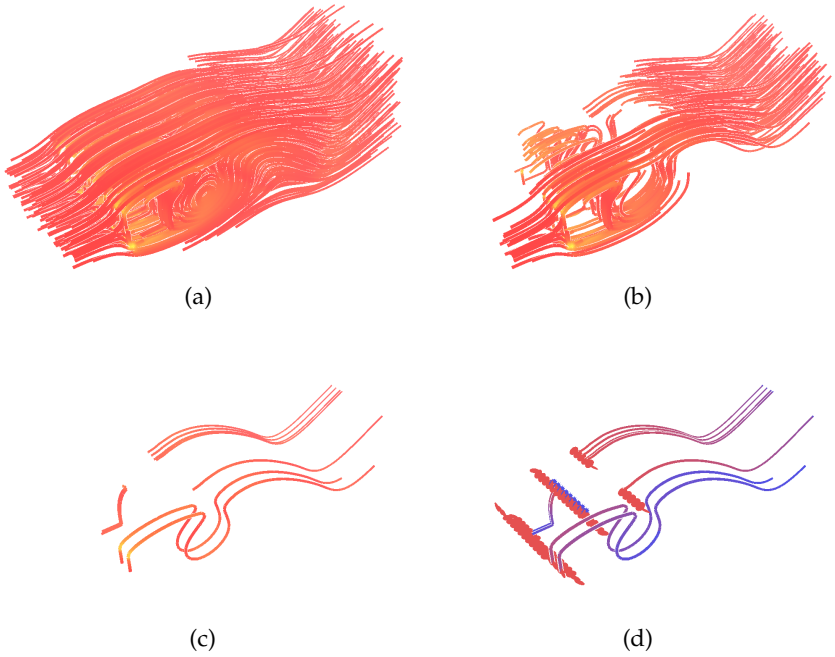


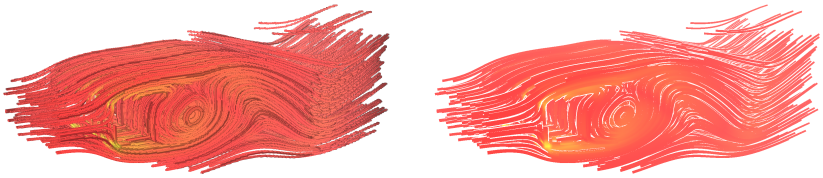
Figure 17: Detailed transition from streamlines to topology: (a) streamlines, (b) thinned out streamlines, (c) topology connectors shown as streamlines, (d) flow topology with topology icons and topology connectors visualized using a dedicated style.

At the beginning of this stage, the saddle connectors still look like streamlines (Figure 17c), but they should look like the saddle connectors introduced by Theisel et al. [2003] (Figure 17d). Therefore, when the structural abstraction increases, the saddle connectors should become wider and orient themselves correctly. At the same time, the icons used to identify the critical points are blended into the visualization. This is done by adding them as small icons at the start of this stage and gradually increasing their size such that they have the proper size at the end of this stage. This makes sure the last transition results in a proper visualization of the topology.

3.2 SUPPORT OF SPATIAL PERCEPTION

In visualizations of both volume and line data, the spatial relations between objects are not always clear, as explained in Section 2.2.2. Because these spatial relations are important when trying to understand the behaviour of the flow, we provide the user the possibility to enhance spatial perception. Common approaches to enhance spatial perception for volume or line visualization are halos [Bruckner and Gröller, 2007; Everts et al., 2009], global shading models [Tarini et al., 2006], or object attenuation [Everts et al., 2009; van der Zwan et al., 2011].

When deciding which techniques should be used to enhance spatial perception we have to take into account that these techniques should not cause any visual discontinuities or artefacts when moving along the structural abstraction axis. Furthermore, we want to have well-defined behaviour everywhere along the spatial perception axis for all stages of structural abstraction, i. e., we should try to avoid regions on the spatial perception axis where no enhancement of spatial perception occurs for any stage of structural abstraction.



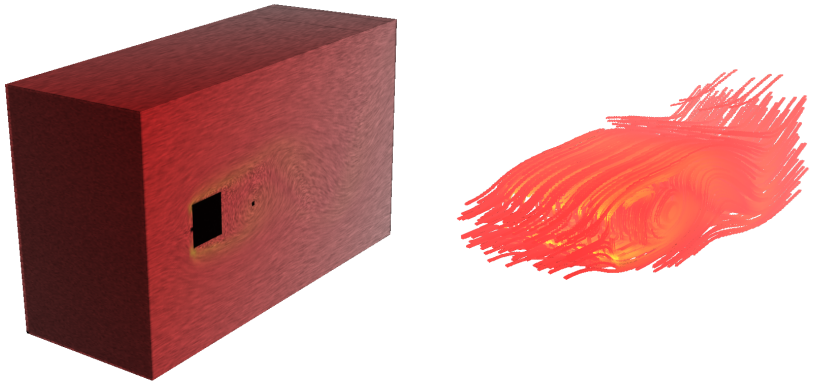
(a) (b)

Figure 18: Spatial perception enhancement using: (a) halos for seedLIC, (b) halos and line attenuation for streamlines.

Halos can be applied to the objects in all stages of structural abstraction, except for the traditional LIC volume, since this block fills the entire domain, the addition of halos would not add to support of spatial enhancement in this case. Therefore, we ignore the halos for the traditional LIC volume. This means, though, that care should be taken to avoid visual artefacts when transitioning from the LIC volume to seedLIC, to which halos can be applied as described by Helgeland and Andreassen [2004] and Interrante and Grosch [1998]. Figure 18 shows the result of applying halos to the seedLIC and streamlines representations.

The second technique we use is object attenuation, the further an object is from the viewer, the smaller it is rendered. This techniques can be applied for line data such as the streamlines (Figure 18b) and saddle connectors and also for the topology icons. However, this technique can not be applied to the tubes of the seedLIC bundles, because their width is determined by the size of the voxels used during computation; to change the width of the seedLIC tubes, we should recompute the seedLIC volume with appropriate voxel sizes. Therefore, we should recompute the seedLIC volume with different voxels sizes whenever the distance to the viewer changes.

For the LIC volume, object attenuation is not a useful technique either, since there is only one object. Therefore, we combine object attenuation with an adaptation of the technique proposed by Boring and Pang [1996] used to highlight vector glyphs. Instead of changing the appearance based on a similarity in direction we use the distance to the viewer as input and make objects which are further away appear dimmer resembling fog as shown in Figure 19.



(a) (b)

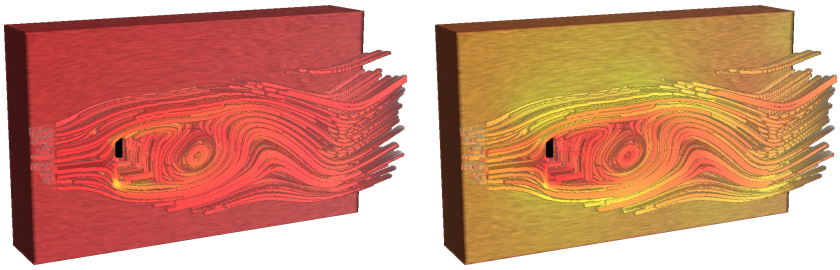
Figure 19: Spatial perception enhancement using fog for: (a) dense LIC volume, (b) seedLIC volume.

We structure the spatial perception techniques along the axis by first applying object attenuation and fog of distance, increasing the effect as the level of spatial perception increases. Along the second part of the axis we apply halos with increasing width. These halos are applied last, because they occlude parts of the images which are further away, making the effects of the other techniques redundant when applied first. Although this means we have a region on the spatial perception axis which does not influence the dense LIC representation, this is not a problem because this does not give visual artefacts and only influences the completely un-abstracted dense LIC volume. When combined with seedLIC, we see the halos being added to the seedLIC tubes as shown in Figure 18a.

3.3 EXTENSION(S)

The abstraction space as described above allows transitions in structure and appearance on a global scale, i. e., for the entire data set. As an extension, we implemented (axis-aligned) clip planes for the traditional LIC volume and let these clip planes interact with the level of structural abstraction. Traditional clip planes allow the inspection of information inside the LIC volume by omitting everything on one side of the plane. Our clip plane does omit all information from the traditional LIC volume, but shows the seedLIC structure instead. By moving the clip plane, the user can study the behaviour in a specific region, visualized by the seedLIC structure while preserving the traditional LIC volume as context, as shown in Figure 20.

When studying the result of flow simulation, researchers are often interested in the magnitude of some physical properties. For example, when designing a wing for a new airplane, the pressure on this wing is of great importance. Therefore, we allow the visualization of physical



(a) (b)

Figure 20: Example of the use of a clip plane revealing the seedLIC volume inside the dense LIC volume using color to represent different physical properties of the flow: (a) colored according to vorticity, (b) colored according to stream magnitude (velocity).

properties in the flow using colors. The user can select between flow magnitude, pressure, and vorticity because these properties are either readily available in the data or can be computed from the input data in a straight-forward manner. Vorticity can be used to determine regions with a lot turbulence which are often of interest to researchers. Based on the magnitude of the user-selected flow property we apply colors to the visualization going from red in regions with minimal magnitude to yellow in regions with maximum magnitude. Figure 20 gives an example of the use of color to indicate physical properties of the flow.

3.4 SUMMARY

Previously, we discussed different techniques which can be used to visualize fluid flow. In this chapter we showed how a selection of these techniques can be used to show the behaviour of the flow at different levels of abstraction and how the different representations can be combined to allow continuous transitions between the different representations. We also discussed which spatial enhancement techniques introduced in Chapter 2 can be applied to the chosen flow representations and how these techniques can be combined along an independent spatial perception axis.

4

REALIZATION

In this chapter we discuss the important aspects of our realization of the abstraction space for fluid flow visualization presented in Chapter 3. We start with discussing the implementation of the different flow representations in Section 4.1. After that, we describe how we realized the transition between these different flow representations in Section 4.2.

4.1 FLOW REPRESENTATIONS

The different flow representations which form the basis of our abstraction space as discussed in Chapter 3 are diverse in their computation and visualization, but also share some features. For example, both LIC and seedLIC result in a volumetric texture which we visualize using the same volume visualization technique and a nearly identical transfer function. This transfer function takes the user-selected flow property into account when determining the color of a point in the flow volume as described in Section 3.3 and is also applied in a modified form to the streamlines.

4.1.1 *LIC*

The original LIC algorithm as introduced by Cabral and Leedom [1993] calculates how the values of a random input texture are moved along the flow, resulting in a texture showing the global flow behaviour. For every voxel x_0 in this texture, the intensity I is given by the convolution integral

$$I(x_0) = \int_{t_0-L}^{t_0+L} k(t-t_0)T(s(t))dt$$

Here, $s(t)$ is the parametrization of the streamline going through $x_0 = s(t_0)$, which is used to find the values along the streamline of the input texture T. The function k is called the *filter kernel* of length $2L$. The kernel can be used, for example, to reduce the influence of points further along the streamline, focusing on local effects.

Computing the convolution integral and the needed streamlines for every voxel in the output texture reveals the behaviour of the flow in the entire flow domain. However, the computed streamlines might cover multiple voxels. Therefore, we use the improved fast-LIC algorithm introduced by Stalling and Hege [1995]. Instead of going

through all voxels and computing the streamline and corresponding convolution integral, fast-LIC updates the intensity value of all voxels which contain part of the streamline. This reduces the amount of streamline calculations, since it is no longer necessary to compute a streamline for every voxel. When a voxel is covered by a sufficient number of streamlines used in the convolution computation for other voxels, no convolution computation is performed for this voxel, further reducing the number of streamline computations.

Besides reducing the number of streamline computations, Stalling and Hege [1995] observe that, for a constant kernel k, the convolution integral of a point x_1 on the streamline close to x_0 differs only by two small correction terms. If we take t_1 such that $s(t_1) = x_1$ and define $\Delta t := t_1 - t_0$ as the distance between t_1 and t_0 (as before, $s(t_0) = x_0$) then:

$$
\begin{aligned}
I(x_1) &= k \int_{t_1-L}^{t_1+L} T(s(t))dt \\
&= k \int_{t_0+\Delta t-L}^{t_0+\Delta t+L} T(s(t))dt \\
&= -k \int_{t_0-L}^{t_0+\Delta t-L} T(s(t))dt + k \int_{t_0-L}^{t_0+L} T(s(t))dt \\
&\quad + k \int_{t_0+L}^{t_0+\Delta t+L} T(s(t))dt \\
&= I(x_0) + k \left[\int_{t_0+L}^{t_0+\Delta t+L} T(s(t))dt - \int_{t_0-L}^{t_0+\Delta t-L} T(s(t))dt \right]
\end{aligned}
$$

For every consecutive point x on the streamline, we can use this relation to determine the intensity $I(x)$ instead of computing the complete convolution integral. Without affecting the visual quality, the integrals can be approximated using Riemann sums [Stalling and Hege, 1995], leading to the following equation for the next point along a discrete streamline representation:

$$I(x_{m+1}) = I(x_m) + k\,[T(x_{m+1+L}) - T(x_{m-L})]$$

where x_i is the position of the i-th point along the discrete streamline and L the length of the discrete filter kernel k. A similar relation can be constructed for points that lie in backward direction along the streamline from the seed point.

The resulting three-dimensional LIC volume is visualized using the hardware-accelerated ray-casting technique mentioned in Section 2.2.1 which we implemented in shaders. Since we deal with a solid block,

only the boundaries of the volume which are defined by the clip planes are visible. While a more simple volume rendering technique would have sufficed, we use ray-casting to allow the transition to the next abstraction stage (sparse seedLIC) as described in Section 4.2.

For the volume visualization we use a multi-dimensional transfer function [Kniss et al., 2001] which takes the position in the flow domain and distance to the viewer as parameters besides the intensity of the rendered texture. We use the HSV color space to construct the final color from the input parameters. The intensity of the input texture controls the value or intensity of the color, revealing the structure of the LIC volume and, therefore, flow behaviour through differences in intensity. The saturation of the color is used to give an extra indication of the structure to the user and to allow for a more fluent transition to the seedLIC volume, where the structure of the volume is less well-defined. The parts of the volume close to the viewer are assigned the highest saturation which decreases when this distance increases, resulting in bright colors close to the viewer and dimmer colors when moving further away.

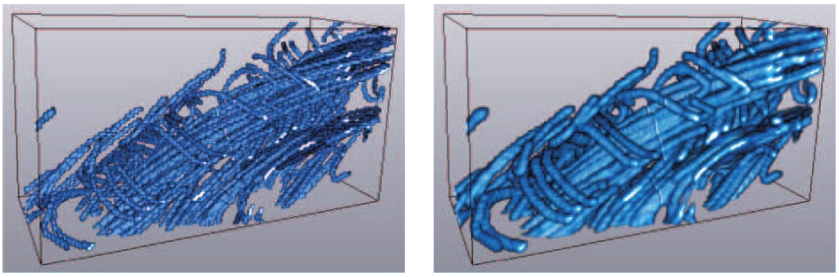
To allow coloring according to a property of the flow, we store these properties in the different channels of a three-dimensional texture. When the user selects a property to use for coloring, this is mapped to the corresponding channel in the texture. Based on the position in the flow domain, the value of the selected flow property can be read from this texture and is used to set the hue of the color, ranging from red for low values to yellow for the highest value occuring in the flow for the selected property.

4.1.2 *SeedLIC*

SeedLIC, the second stage of structural abstraction, provides a more sparse view of the data than the LIC representation used as the unabstracted representation of the data. The sparsity of this representation is a result of using a sparse input texture for the LIC calculation [Interrante and Grosch, 1998; Helgeland and Andreassen, 2004]. This sparse input texture is created by randomly distributing points in a user-defined region of interest based, for example, on the velocity or vorticity magnitude of the flow.

Fast-LIC can be used to compute a sparse LIC visualization based on a sparse input texture as proposed by Interrante and Grosch [1998]. However, Helgeland and Andreassen [2004] remark that during the computation a lot of convolution integrals are computed unnecessarily. Since a sparse input texture contains a lot of regions where all voxels are zero-valued the value of the convolution integral over a streamline in such a region is zero. Therefore, Helgeland and Andreassen [2004] only compute the convolution integral for streamlines calculated from the non-zero points in the input texture which are, therefore, referred

to as seed points. The computation of the intensity uses the same convolution integral as the fast-LIC technique, but is computed only for the seed points and points along the streamlines through the seed points.



(a) (b)

Figure 21: The difference between seedlic without (a) and with (b) applying an isotropic filter.

Restricting the convolution integral computation to streamlines through the seed points results in a sparse representation of the region of interest, which can be computed much faster than the dense fast-LIC representation [Helgeland and Andreassen, 2004]. However, seedLIC only updates those voxels which are covered by streamlines originating from the seed points, in contrast to traditional LIC methods which calculate the intensity for every voxel in the volume. Because voxels next to a streamline are not updated when using seedLIC, aliasing effects can occur, causing the edges of the volume to look jagged as shown in Figure 21a. We use an isotropic filter to smear out the intensity values in the volume and create a smooth results as proposed by Helgeland and Andreassen [2004] and shown in Figure 21b. Smearing out the texture increases the width of the tubes in the texture, which could lead to multiple tubes merging together to one tube which makes it more difficult to see the behaviour of the individual streamlines which are the basis of these tubes.

However, the visual merging of tubes can be reduced by ensuring that there is a minimum distance between the seed points. Increasing the distance between seed points reduces the chance of streamlines getting too close together and, therefore, the chance that the tubes will merge. Our seed point placement algorithm ensures that no seed points are closer together than the minimum distance by checking if there are points within this distance before inserting a seed point into the input texture.

We visualize the resulting seedLIC texture using ray-casting as mentioned before and extend the transfer function used for the dense LIC visualization to work for non-dense data and allow the *limb darkening* halos proposed by Helgeland and Andreassen [2004]. The first extension to the transfer function is that we also include an opacity value, which we set to zero for regions in the texture where the intensity is

close to zero. When combined with a compositing ray function this allows us to see through the empty regions in the texture and reveal those parts of the data that the user identified as relevant. The second modification to the transfer function is that we no longer directly use the intensity of the texture to determine the color value, but apply limb darkening. We do this by introducing a user-defined threshold, which is influenced by the required level of spatial perception enhancement, allowing us to have bigger halos for higher levels of spatial perception enhancement. When the intensity of the texture is above the threshold, we set the color value to one, else we interpolate linearly between zero and one based on the texture intensity. Doing the same for the alpha value results in the limb darkening effect and provides a better perception of the spatial relations in the resulting image [Helgeland and Andreassen, 2004].

4.1.3 *Streamlines*

The streamlines used in the streamline representation are calculated when performing the seedLIC computation. The streamlines computation is started at the seed points and uses a fourth-order Runge-Kutta integration scheme to compute the streamline in both forward and backward direction as proposed by Stalling and Hege [1995] for their fast-LIC technique. The streamline computation is stopped when the length of the streamline has reached a user-defined value, when moving outside the flow domain, or when moving into the geometry which might be present in the simulation data.

A last reason to abort streamline integration is when the flow velocity at the current position is so small that, after several integration steps, we will still be at almost the same position. This situation can result in (nearly) infinite loops while waiting for the streamline to reach the user defined length. Therefore, we keep track of the distance between the last calculated points. When the distance covered during the last integration step is less than a pre-defined threshold, the integration is aborted.

As proposed by Everts et al. [2009] for their depth-dependent halos technique, we use view-aligned triangle strips to render the streamlines. Rendering the streamlines as triangle strips instead of lines has benefits besides allowing the use of depth-dependent halos. We can realize line attenuation by changing the width of the triangle strips based on the distance to the viewer [Everts et al., 2009], which is not possible using line primitives.

To optimize performance, we use a geometry shader to perform the conversion from lines to triangle strips, reducing the amount of data that has to be send to the GPU. The geometry shader is also responsible for calculating the correct width of the triangle strips based on the required level of enhancement of spatial perception;

for line attenuation the width of the strips is reduced based on the distance to the viewer and for the addition of halos, the strips are widened to create a region on the strips on which to draw the halos. Furthermore, we use a modified version of the dense LIC transfer function to determine the color of the streamlines, based only on the position in the flow domain and distance to the viewer for depth-cueing.

4.1.4 *Topology*

The basis of our topology representation are the first-order critical points in the flow. Critical points are points p where the velocity field $\vec{v}$ is zero [Helman and Hesselink, 1989]. When using data resulting from numerical simulation such as the data we use, points where the velocity field is close to zero can also be critical points due to round off effects. Furthermore, a critical point with non-zero Jacobian $\nabla\vec{v}(p)$ is called a first-order critical point.

Theisel et al. [2003] describe how three-dimensional first-order critical points can be classified based on an analysis of the eigenvalues and eigenvectors of the Jacobian $\nabla\vec{v}(p)$. We call the eigenvalues of the Jacobian λ_1, λ_2, and λ_3 and order them according to their real parts, i. e. $Re(\lambda_1) \leqslant Re(\lambda_2) \leqslant Re(\lambda_3)$. Now we can distinguish the following classes of critical points based on the real parts of the eigenvalues:

- *Source*: $0 < Re(\lambda_1) \leqslant Re(\lambda_2) \leqslant Re(\lambda_3)$
- *Repelling saddle*: $Re(\lambda_1) < 0 < Re(\lambda_2) \leqslant Re(\lambda_3)$
- *Attracting saddle*: $Re(\lambda_1) \leqslant Re(\lambda_2) < 0 < Re(\lambda_3)$
- *Sink*: $Re(\lambda_1) \leqslant Re(\lambda_2) \leqslant Re(\lambda_3) < 0$

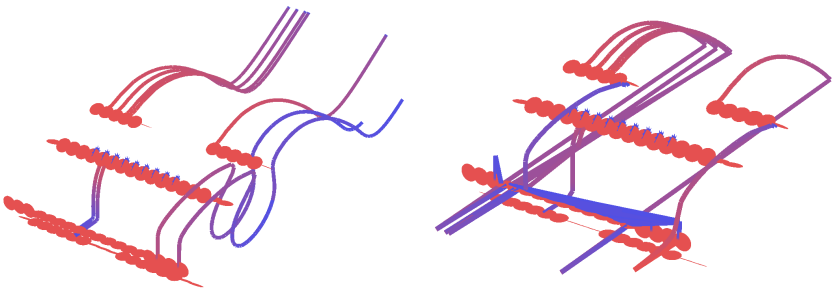
Depending on the presence or absence of imaginary parts in the eigenvalues, we can distinguish two subclasses for each of these classes. When there are no imaginary parts in the eigenvalues for a source, we call it a *repelling node* while we call points for which imaginary parts are present in the eigenvalues *repelling foci*. For the repelling focus there is a plane defined by an eigenvector of the transposed Jacobian matrix on which the point behaves as a two-dimensional repelling focus, while the repelling node shows the same behaviour for all planes. A similar subdivision can be made for sinks, where the points without imaginary parts in the eigenvalues are called *attracting nodes* and the subclass with imaginary parts in the eigenvalues are called *attracting foci*.

A repelling saddle has a plane where repelling behaviour occurs and a direction along which the flow moves towards the repelling saddle point. The behaviour over the plane is determined by the imaginary parts of the eigenvalues. When there are no imaginary parts, the flow

over the plane behaves as a two-dimensional repelling node. Therefore, these critical points are called *repelling node saddle* points. When there are imaginary parts present, the flow along the plane behaves as a two-dimensional repelling focus and the point is called a *repelling focus saddle*. Again, we can apply a similar subdivision for the attracting saddles, where we call the subclass without imaginary parts *attracting node saddles* and the other subclass *attracting focus saddles*.

The saddle points are used to compute the saddle connectors which are defined as the intersection of the separation surfaces of a repelling and an attracting saddle point [Theisel et al., 2003]. Because we are only interested in the intersection of separation surfaces, we do not have to store the entire surface. Instead, we store the front line of the separation surface at the current and the previous integration step. To begin computation we insert a seeding front line around the saddle point. Each integration step we advance the front line by applying forth-order Runge-Kutta integration in the appropriate direction; forward integration when the originating saddle point is a repelling saddle, backward integration otherwise.

After we advance the front lines of the surfaces for the current time-step, we check for intersection of the separation surfaces. Therefore, we perform a triangulation on the front lines and check the resulting triangles for intersections with the line segments of the other front lines. We only check for intersections between separation surfaces originating from opposing saddle points. For example, one originating from a repelling node saddle and the other originating from an attracting focus saddle.



(a) (b)

Figure 22: The result of using different integration directions when computing the saddle connectors: (a) Forward integration from the repelling saddle, (b) Forward integration from the repelling saddle and backward integration from the attracting saddle.

When an intersection between separation surfaces has been found, we know there is a saddle connector between the corresponding saddle points. To find this saddle connector, we have to integrate in the correct direction from both saddle connectors for the same number of integration steps it took to find the intersection. Therefore, we

store a parameter for each point on the front line that allows us to reconstruct the position from which it was seeded. This way, when we find an intersection point, we can compute the saddle connector by starting the computation at the corresponding seeding points.

Because of the underlying physical properties of the flow, the direction in which we perform the integration of the saddle connectors is important. Because we applied backward integration to compute one of the separation surfaces and forward integration to compute the other one, we cannot assume that starting from the seed point for the attracting surface and applying backward integration will result in a point close to the repelling surface. Figure 22 shows the result of using different integration directions to compute the saddle connectors starting in different points and already demonstrates that both integration methods do not give us the intended results as explained in more detail in Chapter 5.

The visualization of the topology consists of two parts, first we draw icons to show the detected saddle points. These icons are drawn on a plane given by the appropriate eigenvectors for the type of critical points as described by Theisel et al. [2003]. We use a texture to show the difference between node and focus saddle points, while we use color to distinguish between repelling (red) and attracting (blue) saddle points. The saddle connectors are drawn as triangle strips similar to the streamlines, to allow both halos and line attenuation.

4.2 STRUCTURAL ABSTRACTION

Now we have discussed how we compute and visualize the different flow representations, we will present how we make the transition between them. The first transition is from traditional dense LIC to seedLIC, which are both volumetric representations of the data. As described above, we use ray-casting to visualize the textures created using these techniques. Instead of using ray-casting to create an image of the two texture and trying to blend between these images, we modified the ray-casting algorithm to blend between the two textures in one pass.

Our ray-casting algorithm results in the correct visualization based on the given level of structural abstraction as well as the clip planes described in Section 3.3. To do this, we modified the ray-casting shader used to visualize the seedLIC texture to keep also track of the projection of the LIC texture. We implement the clip planes by modifying this projection of the LIC texture. Instead of using the value on the border of the LIC volume as defined by the clip planes, we keep moving along the ray until we hit the boundary of the flow domain. While we do this, we blend in the values for the seedLIC volume, revealing the seedLIC structure in the region defined by the clip planes.

The final projection of the two LIC textures is a linear combination of the two projections based on the level of structural abstraction. When no structural abstraction is applied, we use only the dense LIC projection while we use the seedLIC projection at the other end of this transition. When defining the transfer function for our seedLIC visualization, we made sure the empty regions in the seedLIC texture are transparent. The combination of that transfer function with this linear interpolation of the different volume projections results in the dense LIC representation fading away and revealing the seedLIC representation inside when we increase the level of structural abstraction.

We achieve the transition from the seedLIC representation to the streamlines by reducing the opacity of the seedLIC representation while we are already rendering the streamlines. By choosing a line width less then the width of the tubes in the seedLIC visualization, the resulting transition looks like the outer parts of the tubes disappear while the streamlines which are inside remain.

The next transition, the thinning out of the streamlines can also be applied in a straight forward manner. The importance measure used to determine which streamlines we render is computed in a pre-processing step. Therefore, during this transition we only need to determine the importance threshold based on the level of structural abstraction and render the streamlines which have an importance higher than the threshold.

During the last transition we introduce the topology icons and change the appearance of the topology connectors from the style used for the streamlines to a dedicated topology connector style. For the icons we can simply increase the size of the planes on which we render them, starting very small and having the correct size when we reach full structural abstraction. The change of style for the topology connectors is achieved by applying a linear interpolation between the two styles.

4.3 SUMMARY

In this chapter we have discussed our realization of the different flow representations used for our abstraction space as described in Chapter 3. We gave details regarding the computation and visualization of all realizations. Besides describing the realization of the individual parts, we also presented how we realized the transitions between the different representations.

5 RESULTS

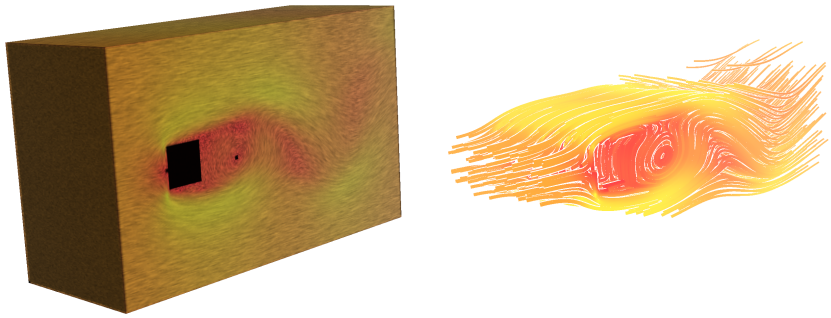
In this chapter we will present results created with our realization of the abstraction space for fluid flow visualization. First, we present example images in Section 5.1, followed by a short discussion on the performance of the program in Section 5.2. Finally, we report informal feedback on our technique in Section 5.3.

We have used two different data sets to create images and measure performance. The first data set show the flow around a cylinder and is computed on a grid with dimensions $100 \times 60 \times 20$. The streamline representation for this data set consists of 953 streamlines and the topology representation contains 75 critical points and 11 connectors. The second data set depicts flow around a block and has dimensions $80 \times 40 \times 40$. For this data set there are 1271 streamlines, 30 critical points and 17 saddle connectors. We created the dense LIC and seedLIC texture at 4 times the dimension of the input data, applying tri-linear interpolation to determine the relevant flow properties on this finer grid to generate images with more detail.

5.1 VISUAL RESULTS

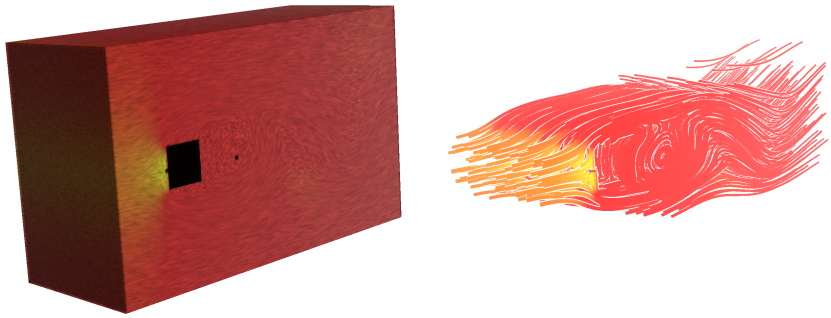
The images in Chapter 3 which show the transitions along the different axes of the abstraction space are all created with our reference implementation. In these images we show the cylinder data set and use color to indicate the amount of vorticity as described in Section 3.3. The user can also choose to use the color to indicate the velocity as shown in Figure 23 or use color to indicate the pressure as demonstrated in Figure 24. While the pressure coloring does not add much information for the used data set because it is only different where the flow hits the cylinder. However, the velocity coloring adds extra information for both the dense LIC volume as well as the streamlines. Without coloring, these representations only show the direction of movement but with the addition of colors indicating velocity we can also see the magnitude of this movement.

Figures 23 and 24 also show the use of the techniques used to enhance spatial perception for both the dense LIC and the streamline representation. In these images we can see the effect of the halos and attenuation for the streamlines and, to a lesser extent, the effect of fog for both representation. Figure 25 shows the use of the techniques to enhance spatial perception for both the seedLIC and the topology representation of the cylinder data set. The effect of the limb darkening halos for the seedLIC volume shown in Figure 25a is very different



(a) (b)

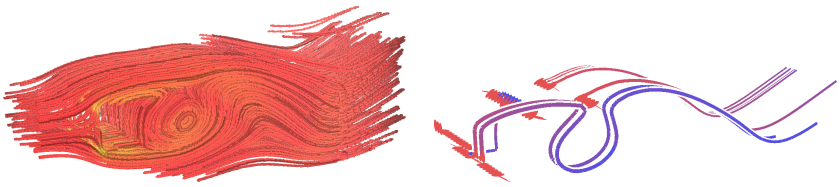
Figure 23: Using color to indicate velocity of the flow in the cylinder data set represented as: (a) dense LIC volume, (b) streamlines.



(a) (b)

Figure 24: Using color to indicate pressure in the cylinder data set represented as: (a) dense LIC volume, (b) streamlines.

from the effect of applying the depth-dependent halos as shown in, for example, Figure 24b. The limb darkening halos emphasize single tubes in the seedLIC volume, while the depth-dependent halos show which streamlines are close together besides emphasizing the streamlines closer to the viewer. This shows that the use of streamlines as representation on our structural abstraction axis is not only appropriate to make the transition from seedLIC to the topology but can enable the user to discover other aspects of the flow through the use of different techniques to enhance spatial perception.



(a) (b)

Figure 25: Applying the techniques used to enhance perception of spatial relations to the cylinder data set represented as: (a) seedLIC volume, (b) flow topology.

The different representations of the flow are placed along the structural abstraction axis in the natural order of increasing level of structural abstraction, with the most detailed representation at the beginning and the most abstract representation at the end. However, for researchers studying a data set the critical points are points where the flow shows special behaviour and identify regions that might need further study. Therefore, a researcher using the abstraction space might start with the topology representation to identify these interesting regions, in this case the flow in the environment of the block (Figure 26a). Then, the level of abstraction can be decreased a bit adding more streamlines which show the behaviour in this region as in Figure 26b. Now it might be interesting to see how the flow behaviour in this small region around the block influences the behaviour of the flow further away from the block, so the level of structural abstraction is decreased further to show more streamlines. At the same time, the coloring is changed to show the velocity and see if this is influenced by the behaviour of the flow around the block (Figure 26c). Finally, the user switches to using seedLIC to show more information of the flow around the block, while at the same time using the clip planes to show the seedLIC volume close to the border of the domain as shown in Figure 26d. The seedLIC volume can now be used, for example, to study if the velocity at this edge of the domain is influenced by the behaviour of the flow around the block.

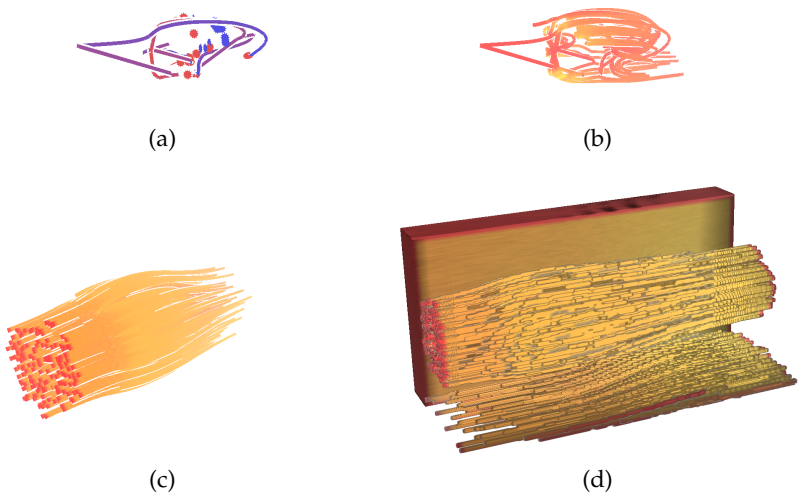


Figure 26: Example use case of using the abstraction space for the block data set: (a) investigate flow behaviour around the critical points using the topology representation, (b) reduce the level of abstraction to show some streamlines as context, (c) show more streamlines and change colors from indicating vorticity to velocity, and (d) finally, show the full seedLIC volume with a part of the dense LIC volume as context.

In the previous example, the last image (Figure 26d) shows an important aspect of flow visualization in general. The seedLIC algorithm also generated tubes near the edge of the domain when using vorticity to determine the region of interest. The vorticity in this region is high due to the choice of boundary conditions for the simulation and would be different for another choice of boundary conditions. Using this simulation data for further computation while assuming the vorticity is only high for the flow around the block could result in strange results which might be unexplainable until the first data set is studied in detail. This example shows that flow visualization should not only be applied to visualize the result but also to verify the outcome of the simulation, preferably during an early stage of the simulation so errors can be corrected without having wasted valuable computation time.

As already indicated in Section 4.1.4, the computation of the saddle connectors in our realization of the topology representation does not give the intended results as presented by Theisel et al. [2003]. One of the problems is indicated in Figure 25b where we can see that the connectors start at a critical point, but then move to regions where no critical points are present. The saddle connectors in Figure 26a are computed using the second method proposed in Section 4.1.4, but this results in two smooth parts of a connector starting at the respective critical points being connected by a straight line. However, for the definition and effectiveness of the abstraction space this does not matter because the incorrect connectors can still be used as basis for the thinning. Since the creation of this abstraction space is the focus of this research and a continuous transition from streamlines to topology is possible using these incorrect saddle connectors, the correction of the computation is left for further research. This research should preferably be conducted in cooperation with numerical mathematicians since the incorrect connectors is probably caused by the used numerical integration methods and the way these methods are applied.

5.2 PERFORMANCE

We tested the performance of our realization by creating a test program which rotates the data around and changes the level of enhancement of spatial perception. Table 1 gives the frame rates achieved for the different representations. When moving between representations, the frame rate did not drop further than the minimum frame rate of the corresponding representations. Therefore, all representations and combinations along the structural abstraction axis can be rendered interactively also when combined with the techniques for the enhancement of spatial perception.

	DENSE LIC	SEEDLIC	STREAMLINES	TOPOLOGY
Cylinder	8 – 13	6 – 9	17 – 20	180 – 260
Block	9 – 15	8 – 11	42 – 46	205 – 290

Table 1: Frame rates (frames per second) of the data sets for the indicated representations of the flow.

The performance measurements are performed on a 2.4 GHz Intel Core 2 Duo with 2 GB memory and an NVIDIA GeForce 320M with 256 MB video memory running on Mac OS X 10.6.8. The window size was approximately 1000×750 pixels of which approximately 75% was covered for the dense LIC visualization.

In Table 1 we can clearly see the benefits of applying abstraction with respect to performance since the frame rates increase with the level of structural abstraction, except for the seedLIC volume. Since we render the full dense LIC volume without using clip planes for the performance measurement of the dense LIC rendering we only need to determine the visualization of the boundary of the dense LIC. However, for the seedLIC representation the algorithm needs to perform more computations since there is no clear surface resulting in a lower frame rate for seedLIC compared to the dense LIC representation.

5.3 INFORMAL FEEDBACK

During the development of the abstraction space we collaborated with researchers from the *Computational Mechanics and Numerical Mathematics* group of the University of Groningen. They provided us with the data sets used to test our realization and gave feedback on both the design of the abstraction space and the realization. The use of colors to show physical properties was a suggestion from one of the mathematicians during an early demonstration.

During an early meeting one of the researchers commented that he liked the topology representation because it showed him everything that was happening in the flow, but that this way of showing the flow is difficult to understand for people unfamiliar with the depicted situation. He continued to suggest that it might be easier for people to understand the abstract topology visualization when they would be able to transition between this representation and a less abstracted representation of the flow. In our opinion this shows that the continuous structural transition is an intuitive way of combining the different representations.

6 DISCUSSION

6.1 CONCLUSION

There exist many different techniques to visualize fluid flow and only a small collection of these methods is covered in Section 2.3. However, the discussed methods already show different levels of abstraction, such as dense, texture-based flow visualization methods which show flow behaviour in the entire domain. Most feature-based flow visualization techniques, however, apply a form of abstraction based on those features of the flow the user is interested in. While the more abstracted visualization techniques improve performance and understanding of the behaviour inside interesting regions, information about the flow in other regions is often lost.

In this thesis we presented a *continuous abstraction space for fluid flow visualization* which can be used to continuously transition between different representations of the flow. Through the continuous change of the level of abstraction and the corresponding continuous transition between representations the user can develop an understanding of the relation between the representations. We showed how the transition between the different representations can be realized and how we can extend the abstraction space to interact with clip planes. In Chapter 5 we gave an example of how the continuous structural abstraction can aid researchers in studying the results of a flow simulation. Furthermore, we discussed different techniques to enhance spatial perception and showed how these techniques interact with the structural abstraction.

6.2 FUTURE WORK

Our current realization is capable of achieving interactive frame rates at all levels of structural abstraction regardless of the level of enhancement of spatial perception. However, the rendering performance for the volume based representations is significantly lower than the performance for the geometry based representations as shown in Table 1. Therefore, the overall frame rate might be improved when the volume renderer algorithm is improved, for instance by subdividing the seedLIC volume and only rendering non-empty regions.

A downside of the current realization is the amount of pre-processing time required to compute all different representations of the flow. Reducing the time it takes to compute the representations will allow for easier fine-tuning of the parameters of the different representations.

For example, when we change the region of interest of the seedLIC representation for our example data set, it takes from five minutes to half an hour to compute the new representation, depending on the size of the region of interest and the required resolution of the result. Computing the topology connectors and the dense LIC block also takes a long time which might be reduced, for example, by using the computing power of the graphics hardware to compute these representations or parts of them.

There are also possible enhancements that can be made to our developed abstraction space. We selected the various used flow visualization techniques because, together, they allow for the continuous selection of the level of abstraction, but different flow representations such as stream surfaces might work just as well or perhaps even better. Stream surfaces cannot be integrated in the current abstraction space in a straight-forward way but a solution is to allow the user to select a structural abstraction path. One path would be our current structural abstraction axis while the second path also starts at the dense LIC representations but includes stream surfaces and perhaps other representations related to stream surfaces.

Furthermore, our current abstraction space only considers time-independent data while researchers are also interested in time-dependent data. The extension of the current abstraction space might help studying time-dependent data. However, this is not a straight-forward extension because the user should be able to distinguish which animation comes from the time-dependence and which animation is part of a transition in representation. Besides the challenges of visualizing the three-dimensional data, this is another example of the importance of optimizing the calculation of the separated representations.

BIBLIOGRAPHY

Arthur Appel, F. James Rohlf, and Arthur J. Stein. The haloed line effect for hidden line elimination. In *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 151–157, New York, NY, USA, 1979. ACM.

David C. Banks and Bart A. Singer. Vortex tubes in turbulent flows: identification, representation, reconstruction. In *Proceedings of the conference on Visualization '94*, VIS '94, pages 132–139, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

G.K. Batchelor. *An introduction to fluid dynamics*. Cambridge University Press, 1967.

Ed Boring and Alex Pang. Directional flow visualization of vector fields. In *Proceedings of the 7th conference on Visualization '96*, VIS '96, pages 389–392, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

Stefan Bruckner and Eduard Gröller. Enhancing depth-perception with flexible volumetric halos. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1344–1351, 2007.

B. Cabral and L.C. Leedom. Imaging vector fields using line integral convolution. *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270, 1993.

Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*, VVS '94, pages 91–98, New York, NY, USA, 1994. ACM.

A. Chorin. Numerical Solution of the Navier-Stokes Equations. *Mathematics of Computation*, 22:745–762, 1968.

T. J. Cullip and U. Neumann. Accelerating volume reconstruction with 3d texture hardware. *Tech. Rep. TR93-027*, 1993.

Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003.

G. Elber. Line illustrations $\in$ computer graphics. *The Visual Computer*, 11(6):290–296, 1995.

Maarten H. Everts, Henk Bekker, Jos B.T.M. Roerdink, and Tobias Isenberg. Depth-dependent halos: Illustrative rendering of dense

line data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1299–1306, 2009.

L.K. Forssell and S.D. Cohen. Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *Visualization and Computer Graphics, IEEE Transactions on*, 1(2):133–141, 1995.

V. Girault and P. Raviart. *Finite element methods for Navier-Stokes equations: Theory and algorithms*, volume 5. 1986.

A. Globus, C. Levit, and T. Lasinski. A tool for visualizing the topology of three-dimensional vector fields. In *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 33–40, 408, oct 1991.

Bruce Gooch and Amy Gooch. *Non-photorealistic rendering*. AK Peters, Ltd., 2001.

Markus Hadwiger, Christian Sigg, Henning Scharsach, Khatja Bühler, and Markus Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

Helwig Hauser, Robert S. Laramee, Helmut Doleisch, Frits H. Post, and Benjamin Vrolijk. The state of the art in flow visualization, part 1: Direct, texture-based, and geometric techniques. Technical report, 2003.

A. Helgeland and O. Andreassen. Visualization of vector fields using seed lic and volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):673–682, 2004.

James L. Helman and Lambertus Hesselink. Representation and display of vector field topology in fluid flow data sets. *Computer*, 22: 27–36, August 1989.

J.L. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, pages 36–46, 1991.

J.P.M. Hultquist. Constructing stream surfaces in steady 3D vector fields. *Proceedings of the 3rd conference on Visualization'92*, pages 171–178, 1992.

V. Interrante and C. Grosch. Visualizing 3d flow. *Computer Graphics and Applications, IEEE*, 18(4):49 –53, jul/aug 1998.

S. Islam, D. Silver, and Min Chen. Volume splitting and its applications. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):193–203, march-april 2007.

B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. *Visualization in Scientific Computing*, 97:43–56, 1997.

B. Jobard and W. Lefer. Multiresolution flow visualization. *WSCG'01, Plzen, Czech Republic*, 2001.

James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, 1986.

Arie Kaufman and Klaus Mueller. *Overview of Volume Rendering*, chapter 7, pages 127–174. Academic Press, first edition, 2005.

Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 255–262, Washington, DC, USA, 2001. IEEE Computer Society.

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.

M. Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8(3):29 –37, may 1988.

Zhanping Liu and Robert J. Moorhead. Accelerated unsteady flow line integral convolution. *IEEE Transactions on Visualization and Computer Graphics*, 11:113–125, March 2005.

O. Mattausch, T. Theußl, H. Hauser, and E. Gröller. Strategies for interactive exploration of 3d flow using evenly-spaced illuminated streamlines. *Proceedings of the 19th spring conference on Computer graphics*, pages 213–222, 2003.

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.

Marc Nienhaus and Jürgen Döllner. Sketchy drawings. In *AFRIGRAPH '04: Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, pages 73–81, New York, NY, USA, 2004. ACM.

S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 5(3):238 –250, jul-sep 1999.

D. Patel, C. Giertsen, J. Thurmond, J. Gjelberg, and E. Gröller. The seismic analyzer: Interpreting and illustrating 2d seismic data. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1571 –1578, nov.-dec. 2008.

Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18:311–317, June 1975.

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.

Peter Rautek, Stefan Bruckner, Eduard Gröller, and Ivan Viola. Illustrative visualization: new technology or useless tautology? *SIGGRAPH Comput. Graph.*, 42:4:1–4:8, August 2008.

C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive exploration of volume line integral convolution based on 3d-texture mapping. In *Proceedings of the conference on Visualization '99: celebrating ten years*, VIS '99, pages 233–240, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

I.A. Sadarjoen and F.H. Post. Geometric methods for vortex detection. In *Data Visualization '99. Proc. Vis-Sym'99*, pages 53–62, 1999.

Martin Schulz, Frank Reck, Wolf Bartelheimer, and Thomas Ertl. Interactive visualization of fluid dynamics simulations in locally refined cartesian grids (case study). In *Proceedings of the conference on Visualization '99: celebrating ten years*, VIS '99, pages 413–416, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

Detlev Stalling and Hans-Christian Hege. Fast and resolution independent line integral convolution. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 249–256, New York, NY, USA, 1995. ACM.

Thomas Strothotte and Stefan Schlechtweg. *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann, 2002.

Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

A. Telea and JJ van Wijk. 3D IBFV: hardware-accelerated 3D flow visualization. *Visualization, 2003. VIS 2003. IEEE*, pages 233–240, 2003.

H. Theisel, T. Weinkauf, H.C. Hege, and H.P. Seidel. Saddle connectors-an approach to visualizing the topological skeleton of complex 3D vector fields. *Visualization, 2003. VIS 2003. IEEE*, pages 225–232, 2003.

Xavier Tricoche, Gerik Scheuermann, and Hans Hagen. Continuous topology simplification of planar vector fields. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 159–166, Washington, DC, USA, 2001. IEEE Computer Society.

Matthew van der Zwan, Wouter Lueks, Henk Bekker, and Tobias Isenberg. Illustrative molecular visualization with continuous abstraction. *Computer Graphics Forum*, 30(3):683–690, 2011.

Jarke J. van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 309–318, New York, NY, USA, 1991. ACM.

J.J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics (TOG)*, 21(3):745–754, 2002.

J.J. van Wijk. Image Based Flow Visualization for Curved Surfaces. *IEEE Visualization: Proceedings of the 14 th IEEE Visualization 2003(VIS'03)*, 2003.

W. von Funck, T. Weinkauf, H. Theisel, and H.P. Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. | *IEEE Transactions on Visualization and Computer Graphics*, pages 1396–1403, 2008.

J.W. Wallis, T.R. Miller, C.A. Lerner, and E.C. Kleerup. Three-dimensional display in nuclear medicine. *Medical Imaging, IEEE Transactions on*, 8(4):297–230, dec 1989.

Joseph R. Weber. ProteinShader: Illustrative Rendering of Macromolecules. *BMC Structural Biology 2009*, 9(19):1–19, May 2009.

Qi Zhang, Roy Eagleson, and Terry Peters. Volume visualization: A technical overview with a focus on medical applications. *Journal of Digital Imaging*, 24:640–664, 2011. 10.1007/s10278-010-9321-6.

Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. In *Proceedings of the 7th conference on Visualization '96*, VIS '96, pages 107–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.