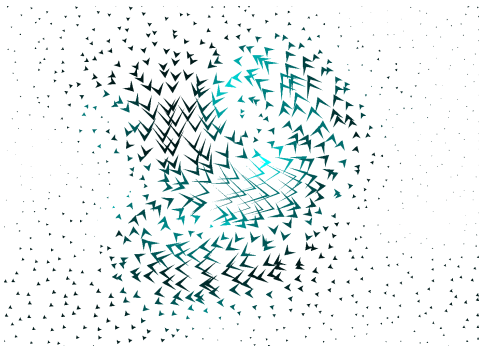
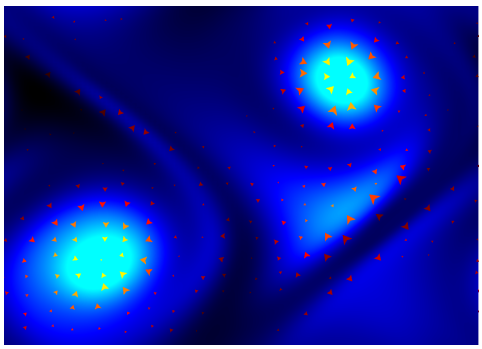
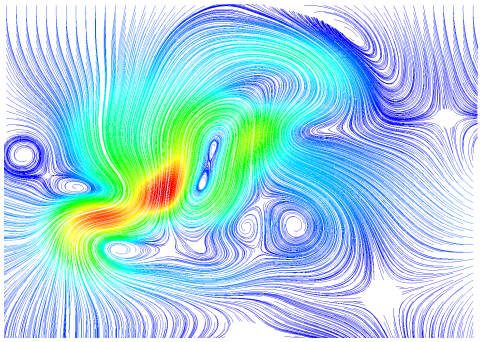
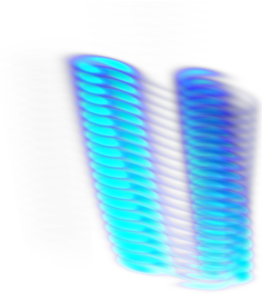
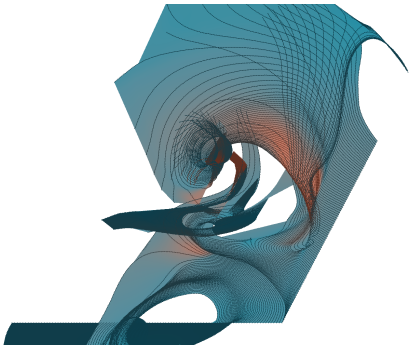
Scientific Visualization

An Interactive Application for Real-Time Simulation of Fluid Flow.

L.E.N. Baakman s1869140
S.J. van Loon s1795813

March 2, 2018

Contents

Introduction

This document accompanies the visualization application made for the course Scientific Visualization at the University of Groningen. The application provides different methods to visualize a real-time fluid flow simulation. This simulation calculates the values of quantities such as fluid density and fluid velocity over time. In chapter 1 the simulation is shortly discussed as well as the techniques used to develop the simulation.

Next chapter 2 focuses on the visualization of scalars with colors. Chapter 3 and chapter 5 are concerned with the visualization of time-independent vector fields, by way of glyphs and streamlines, respectively. In chapter 4 new vector fields, namely the gradient fields are introduced, and visualized with glyphs.

Lastly chapter 6 and chapter 7 discuss the visualization of time-dependent vector fields, with slices and stream surfaces, respectively.

Chapter 1

Fluid Visualization

In this chapter the skeleton code of the visualization project is introduced. The skeleton code contains a real-time fluid simulation plus some supporting functionality which takes care of the interaction between the simulation and the rest of the application. The purpose of our application is to provide different visualization methods to give a visual representation of these variables in order for the user to obtain insight about the fluid dynamics in the simulation.

As the focus of this project is on the visualization of the simulation, and not on the how and why of the simulation we treat the simulation as a black box. In section 1.1 we shortly introduce the output of this black box, section 1.2 presents the input we can provide to it.

1.1 Fluid Simulation

The fluid simulation used in this project is developed by Stam [3]. The simulation computes the velocity and the density of the fluid based on the user-supplied input force. The simulation is sampled on an uniform 50×50 grid that loops around, e.g. fluids 'disappearing' at one edge of the grid are added at the opposite edge.

The simulation contains one scalar field and two vector fields: the fluid density (ρ), the fluid velocity ($\mathbf{v}$), and the force field $\mathbf{f}$. Two logical derived scalar fields are the fluid velocity magnitude, $||\mathbf{v}||$ and the force field magnitude, $||\mathbf{f}||$.

1.2 Simulation Interaction

We allow a number of inputs to the black box that is the simulation. The most important of which is adding force somewhere within the domain of the simulation. This process of adding force is akin to steering a fluid. The user of the application can add the force via interaction with the mouse. The amount of force added a the position of the mouse movement can be set interactively.

Other than the force the user can also control the speed of the simulation interactively. Alternatively the simulation can be paused, which is comparable to instantly freezing the fluid. In this frozen state one can move to the next state of the simulation.

Chapter 2

Color Mapping

Color mapping is widely used for the visualization of scalar data. This is done by associating a color with each scalar value that is visualized. In this chapter we will discuss the color maps included in the application and how these color maps are supported. In section 2.1 the general method used for color mapping is first explained as well as the parametrization and application of color maps. Next Section 2.2 highlights the important features of color maps and discusses the color maps that are implemented in our application. For each color map that is provided in our application its advantages and disadvantages are mentioned, as well as a small motivation for its inclusion. Finally, section 2.2.6 shows the results of applying color maps to the simulation and discusses these results.

2.1 Method

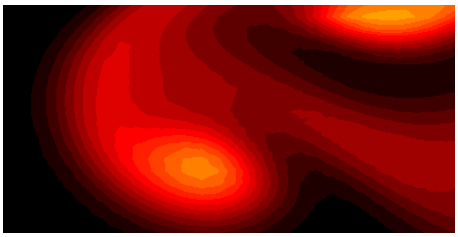
In this section the main methods for color mapping are discussed. In section 2.1.1 we discuss *texture mapping*, the method used to map scalar values to colors. Next section 2.1.2 considers the parametrization of color maps and explains how the number of colors and the saturation can influence the quality of a color map. Last section 2.1.3 shows how color maps are applied to scalar data.

2.1.1 Texture Mapping

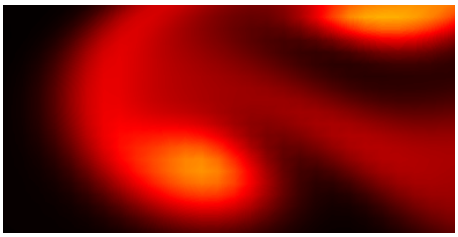
To implement the color maps *texture mapping* is used. The general pipeline of texture mapping is as follows. First the scalar values are sampled for each vertex in the simulation grid. Next the vertices of the triangulation of the simulation grid are passed to the vertex shader with the associated scalar values. The vertex shader normalizes scalar values associated with the vertices using a given range of the scalar values creating texture-coordinates for every vertex. This step ensures that the texture coordinates are in the range [0, 1]. The texture-coordinates are passed to the fragment shader and are automatically interpolated by OpenGL. The fragment shaders retrieves the color that is associated with the texture coordinate it receives.

The end result is a piecewise-linear reconstruction of the scalar field. By interpolating the texture coordinates, interpolation of colors is prevented. This ensures that no false colors are displayed in the visualization. This is also the main motivation for the choice of texture mapping. The alternative, naive, method of vertex-based color mapping can result in colors that do not correspond with the actual scalar values or worse, colors that are not in the color map. These problems are prevented by avoiding the interpolation of colors.

Since we use texture mapping a color map is defined as a 1D texture that represents the range o colors that are associated with the scalar values normalized to the range [0, 1]. Defining color maps as 1D image, has the added advantage of making the definition of new color maps trivial, as evidenced by the large number of color maps supported by our application.



(a) *Example of a visualization using a heat-map containing 20 colors.*



(b) *Example of a visualization using a heat-map containing 256 colors.*

Figure 2.1: *The influence of the number of colors in the color map shown, using the heat-map using (a) 20 and (b) 256 colors. Both images are taken from the same simulation state and zoomed in on the same location.*

2.1.2 Parameterization of Color Maps

Given a color map scheme there are two parameters which can influence the final map. The number of colors in the color map and the saturation of the colors, these parameters are discussed in section 2.1.2 and section 2.1.2, respectively.

Number of Colors

The number of colors influences how smooth the color map is perceived to be. When the number of colors in a color map is low, there is a relatively large difference between the scalar values associated with two neighboring colors in the color map. In the visualization this large step shows up as sharp edges between areas with different scalar values. An example of this banding effect is given in figure 2.1 which shows the difference between two color maps that use the same color scheme, but a different number of colors. We observe that figure 2.1a shows distinct bands of colors, whereas figure 2.1b has smooth transitions between the colors. In our application the user can select the desired number of colors form the range [2, 256]. The default value is set to 256 colors to ensure a smooth color map. Using a color map with fewer colors would introduce sharp transitions in the visualization that are not present in the data.

Saturation

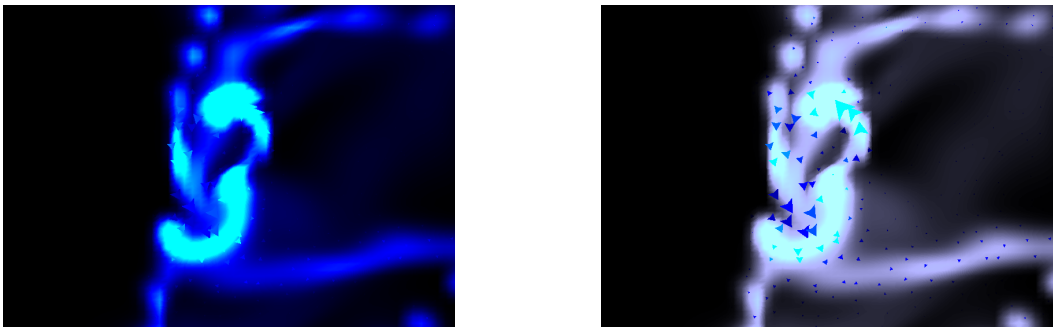
Besides the number of colors it is also possible to change the saturation of a subset of the color maps. Saturation influences the intensity of a color: colors with full saturation will appear pure, whereas colors with low saturation appear more washed out. Changing the saturation of a color map can be of use when displaying multiple visualization techniques together. An example of the effect of changing saturation is given in figure 2.2. In figure 2.2a the glyphs are almost non distinguishable from the scalar visualization, their visibility is greatly enhanced in figure 2.2b, which uses a color map with a lower saturation. In the application it is possible to change the saturation of most color maps, however the user should keep in mind that lowering the saturation can cause a less distinct ordering of the colors in the color map.

2.1.3 Applying Colormaps

An important aspect of using color maps is how they are applied to the scalar values. Given a color map of N colors with colors $c_1, c_2, \cdots, c_N$ and scalars in the range $[f_{\min}, f_{\max}]$ each scalar values is mapped to a color via linear scaling. The choice of $[f_{\min}, f_{\max}]$ is important when using a color map. Imagine that we have picked $f_{\min} = 0$ and $f_{\max} = 100$ but the actual range of the scalar values in the simulation is between 10 and 20. In this case the visualization would only use one out of ten colors present in the color map. Ideally the entire range of the color map is applied to the full range of scalar values.

In our application $[f_{\min}, f_{\max}]$ can be set in two ways. The first method uses a fixed range of $f_{\min} = 0$ and picks $f_{\max}$ based on the scalar value visualized and the forces that are input by the user[1]. These $[f_{\min}, f_{\max}]$

[1] Given an input force set to 10 we have (empirically) determined that the fluid density has $f_{\max} = 10$, fluid velocity magnitude has



(a) *scalar visualization saturation = 1.0* **(b)** *scalar visualization saturation = 0.30*

Figure 2.2: *An illustration of the influence of saturation when combining scalar visualization and vector visualization. The vector visualization is fully saturated in both images, the color map used for the scalar visualization is (a) fully saturated, and (b) partially saturated.*

are initialized as such, however the user can update these fixed ranges based on the current dynamic range. These fixed ranges allow the user to observe the change in the scalar field as a function of time. The disadvantage is that after some time in which no new forces are added to the simulation, the range of the scalar values only uses a small subset of the color map. Which makes it hard to observe the differences between scalar values.

The second method uses a dynamic range $[f_{\min}, f_{\max}]$. This is done by determining the minimum and maximum of the all scalar fields for every time-step in the simulation and updating $[f_{\min}, f_{\max}]$ accordingly. The advantage of this approach is that at every time-step the full range of the color map is used. But it hides the changes of the scalar field as s function of time.

Clamping

Previously we assumed that the scalar values were uniformly distributed. However, some simulations might result in non-uniformly distributed scalar values, e.g. most scalar values fall within a range much smaller than $[f_{\min}, f_{\max}]$. In these cases it might be desirable to show more detail in the range that contains the most scalar values. To facilitate this the user can adapt $[f_{\min}, f_{\max}]$ to a range $[f_{min}^{clamp}, f_{max}^{clamp}]$ such that $f_{\min} \leq f_{min}^{clamp}$ and $f_{\max} \geq f_{max}^{clamp}$[2]. Scalar values outside the range $[f_{min}^{clamp}, f_{max}^{clamp}]$ are set to the extreme colors of the color map. This has as disadvantage that it is not possible to distinguish between scalar values outside of the clamping range, but offers better resolution inside this range. An example of clamping is given in figure 2.3, here the clamped variant shows much more detail but shows the same color for all scalar values between 0.1 and 10.

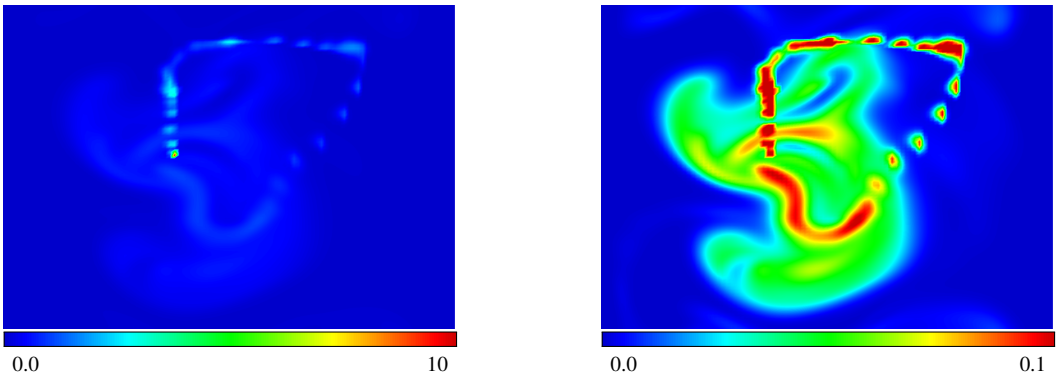
2.2 Provided Color Maps

This section discusses the color maps provided in the application. Each color map is accompanied with a short discussion of its advantages, disadvantages and in which situations it is best applied.

As discussed in the previous section, color maps provide a way to map scalar values to colors in order for the user to extract information. The quality of a color map can be judged by how well and intuitive the color map allows inverse mapping. Which color map is best depends on the context and the goal of the visualization. For example the zebra color map is well suited to the visualization of scalar values that vary quickly, i.e. the first derivative, but is less useful if the aim was to highlight maximum values. Historical reasons can also influence the choice of color maps. For example X-rays are for example always displayed with a gray-scale color map, in spite of the existence of alternative color maps.

$f_{\max} = 0.1$, and the force field magnitude has $f_{\max} = 0.1$.

[2] At all times it is ensured that $f_{max}^{clamp} > f_{min}^{clamp}$



(a) *no clamping*

(b) $[f_{min}^{clamp}, f_{max}^{clamp}] = [0 \cdot f_{min}, 0.1 \cdot f_{max}]$

Figure 2.3: *Example of a visualization (a) without and (b) with clamping. The clamped version shows more detail, but does not distinguish scalar values above* 0.1. $[f_{min}, f_{max}] = [0, 10]$.

In our application we have chosen to implement a variety of color maps, this way the application offers a suitable color map for almost any form of visualization, ranging from the (in)famous rainbow color map to luminance based, two-hue and diverging color maps.

2.2.1 Rainbow Colormap

The rainbow color map, illustrated in figure 2.4a is one of the most used color maps in the scientific community and is therefore included in our application. The color map is constructed by first doing a piecewise interpolation of blue to green and next from green to red. The resulting colors are a mix of blue and green or a mix of green and red, except for the tails of the color map which contain only blue or red.

The color map is often seen as visually appealing due to the varying colors and can give an intuitively mapping of high and low values to warm (red) and cold (blue) colors. However, although often used, the rainbow color map has many flaws and is heavily critiqued[1, 2]. Some of the main disadvantages of this color map are discussed below.

- It pulls the focus of its users towards areas with high values. This can be an advantage when these are the values of interest, but it might distract otherwise.
- All colors in the color map have the same luminance, except for its most extreme colors, but many

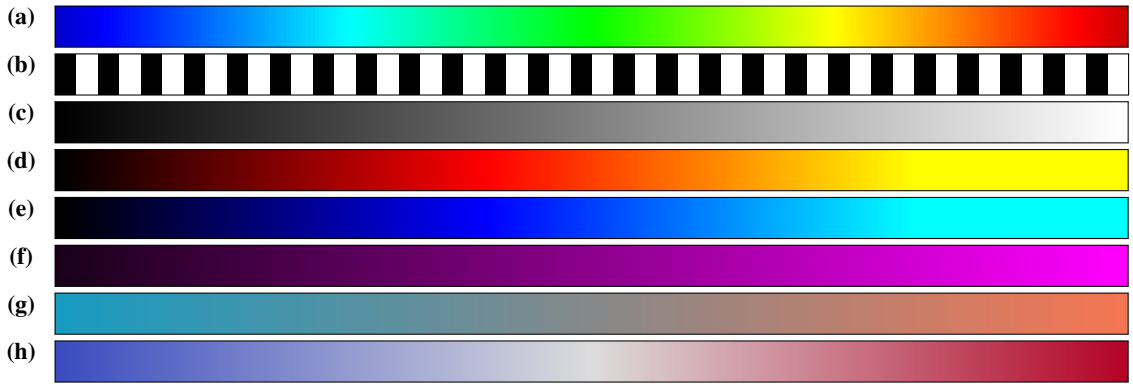


Figure 2.4: *Color maps provided in the application: ((a)) rainbow, (b) zebra, (c) gray-scale, (d) heat, (e) cold, (f) user picked hue, (g) linear two-color, and (h) blue-red diverging. Each color map is shown using full saturation and 256 colors, except for the zebra color map which has 50 'colors'.*

perceive the colors in this color map to have different luminosity. As a result the ordering of the luminance in the colormap might appear to be non-monotonic, which can confuse the inverse mapping process.

- There is no decisive ordering of the colors in this color map. It orders the colors from blue, to green, to yellow, to red, which is not a order naturally perceived.
- The color map does not behave well when the saturation is lowered or when translated to black and white, for example when printing in black and white. Doing so results in a broken, non-monotonic color map, which disallows proper inverse mapping.
- It is not is not colorblind friendly.

2.2.2 Luminance Color maps

The next set of color maps use luminance to give a natural ordering of the colors (and thus values). These color maps are often effective since human perception is sensitive to changes in luminance[2] which makes inverse mapping easier. Luminance based color maps have three major disadvantages. Firstly they are not well-suited for use in combination with 3D shading, since shading can cause changes in the perceived luminance of a color or vice verse changes in luminance might give the illusion of shading, where there is none. Secondly luminance based color maps draw attention to areas with high luminance, details in low luminosity areas are often hard to see and are therefore likely to be overlooked. A third flaw is that pixels with the same luminance might look different depending on the luminance of the surrounding pixels. We support for different luminance color maps, which are discussed in sections 2.2.2 to 2.2.2.

Gray-Scale color map

The gray-scale color map is given shown figure 2.4c is the simplest of the luminance based color maps and does not use any hue. The advantage of this color map is that it works well for a lot of media and does not rely on changes in hues for inverse mapping. This makes this color map suitable for many applications and users. Besides the general disadvantages associated with luminance based color maps, this specific color map is not well suited to non compact datasets, e.g. datasets containing holes. In these cases the background might be similar to objects in the dataset, making them blend into the background.

Single Hue Color map

The single hue color map is an extension on the gray-scale color map. This color map is created by adding a single hue to the gray-scale color map. This can make the color map more visual appealing and removes the probability of the visualization blending in with the background. In figure 2.4f an example is shown, using a pink hue.

Heat Colormap

The heat map is a luminance based color map that uses both luminance and a range of hues for the ordering and thus provides more clues as to the ordering of the colors, compared to the gray-scale and single hue color map. Although it uses less hues than the rainbow color map it has as the advantage that the used hues have a natural perceived ordering, namely red, to orange, to yellow.

In some applications the heat color map uses white for the highest value data instead of the yellow we use. Since our application has a white background, we removed the white value from the color map. The color map is displayed in figure 2.4d.

Cold Color map

The cold color map is similar to the heat color map except it uses blue tinted colors instead of red tints. It is added since blue and red are nice contrasting colors making it possible to combine scalar visualization with

glyphs by using the heat map for one visualization and the cold color map for the second visualization. The color map is displayed in figure 2.4e.

2.2.3 Isoluminant Color map

The isoluminant color map, shown in figure 2.4g, is a cyan-to-mauve color map based on linear interpolation[2]. Because the color map is isoluminant it is well suited to be used in combination with 3D shading since differences in luminance can only be caused by shading. Opposed to the green-to-red isoluminant color map that is also often used, the cyan-to-mauve color map is colorblind friendly, making it a more suitable candidate. Although well suited for use in 3D visualizations the color map has two major flaws. The human perception is less sensitive to changes in hue than in changes in luminance, making the inverse mapping processes of this color map harder than it is for luminance based color maps. Furthermore, since the color map offers less color variation than a lot of other color maps it is difficult to visualize a large range of values, the colors of the resulting visualization are often perceived as dull.

2.2.4 Diverging Color map

The diverging color map is constructed by the interpolation of two hues at the end of the color map with a middle hue. In other words, given three colors c_{min}, c_{mid}, and c_{max} the color map is constructed by first interpolating c_{min} and c_{mid} and next interpolating c_{mid} and c_{max}. In our application the three colors are taken from [2], which proposes a blue tint for c_{min}, an off-white for c_{mid} and a red tint for c_{max}. The resulting color map is shown in figure 2.4h.

The diverging color map is mostly used in cases the median value is of importance. By the interpolating three colors the areas where the scalars approximate the median value are clearly distinguishable by their white color. Furthermore there is a clear separation between scalars greater than and smaller than the median value, the first are shown in red, the second in blue. A disadvantage of the diverging color map is that there is no clear natural ordering of the colors. Furthermore separation of the scalars by their median value is not always relevant.

2.2.5 Zebra Color map

The zebra color map differs significantly from the previously discussed color maps. It only uses two alternating colors and thus does not offer an inverse mapping from color to scalar value. However, the color map is very useful for showing variations in the data. It can be used to detect regions in which data changes quickly or alternatively, where data stays constant. When applying the color map to the data, care should be taken that the number of colors is not too high. When this is the case the black/white bands might vary too quickly and blend together into a gray area and thus results in false colors. figure 2.4b gives an example of the color map, with 50 color bands.

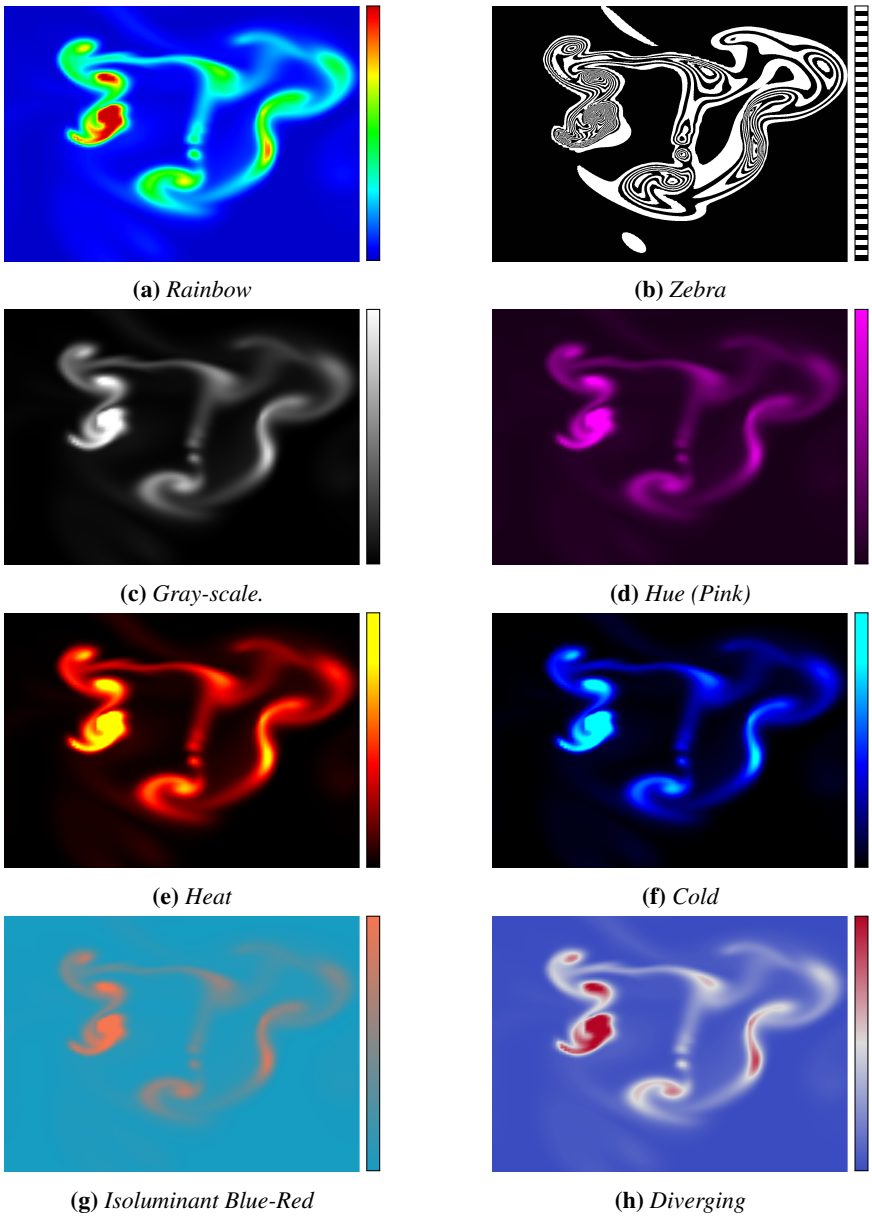
2.2.6 Results

In figure 2.5 a visualization from simulation snapshot is shown using all the different color maps available in the application. Comparing these figures we observe firstly that in the visualization with the rainbow color map, figure 2.5a, the maximum values are very prominent and that there is a clear distinction between the blue, green, and green areas but no clear transition between those areas. This fits with the disadvantages discussed in section 2.2.1. Figure 2.5b indeed illustrates the areas with high and low variation.

Comparing the luminance based color maps, figures 2.5c to 2.5f, we observe in all cases a nice transition and from areas with low to areas with high scalar values. Also note the added benefit of the maps containing hues compared to the gray-scale color map; especially in the areas containing low values a bit more variation is visible.

The isoluminant blue-red color map, shown in figure 2.5g, does not offer as much details as the other color maps. This confirms that these color maps are not well suited for 2D visualizations.

In the diverging color map, illustrated in figure 2.5h, one easily recognizes the same distinct areas shown by the rainbow color map. Compared to the rainbow color map, there are two distinct differences.



(a) *Rainbow*

(b) *Zebra*

(c) *Gray-scale.*

(d) *Hue (Pink)*

(e) *Heat*

(f) *Cold*

(g) *Isoluminant Blue-Red*

(h) *Diverging*

Figure 2.5: *A visualization of the fluid density (ρ) using the color maps available in the application. All color maps uses 256 colors except for the zebra color map which uses 50. All color maps are fully saturated.*

First the maximum values do not draw as much attention as they do in the rainbow color map, furthermore we can see a more natural transition from low to high values making this color map more suitable for visualization of these data.

Chapter 3

Glyphs

In this chapter a set of *Glyphs* is introduced. Glyphs are a visualization technique which can be used to visualize a vector field by showing its orientation and magnitude. This is done by using icons that show direction and magnitude. A disadvantage of glyphs is the size needed for the icons; as they need to show both direction and magnitude they need to be of some size to convey this information in a clear matter. This means that the number of glyphs used in the visualization needs to be restricted to avoid clutter and glyphs blending together. This means the visualization can not be as 'dense' as the scalar visualization, consequently it is not possible to show the information we have for every vertex of the simulation grid. In the next section we discuss how the simulation is sampled and how we prevent cluttering. In section 3.2 the different glyphs we have chosen to implement are discussed.

3.1 Method

The main method for constructing glyphs is to sample the dataset and place a glyph at every sample-point. These glyphs are then scaled, colored and rotated based on the the vector and scalar field that are used.

3.1.1 Visualization Grid

Since glyphs can not be shown for every vertex of the simulation due to their size, we need to subsample the simulation. The number of sampling points denotes how many glyphs are shown. Fewer sampling points result in fewer glyphs and thus less clutter, but also less displayed information. Increasing the number of sampling points increases the amount of information shown, but is also likely to increase cluttering which can interfere with how well the information can be understood by the user.

To facilitate the sampling, a visualization grid is constructed[1]. This grid is defined on top of the simulation grid, ensuring that simulation is covered. Each vertex of the visualization grid is contained within some cell of the simulation grid. When data is requested from vertices in the visualization grid the values at that vertex are calculated using bilinear interpolation of the vertices of its containing cell in the simulation cell.

Given a cell C with four vertices: the upper left vertex (C_{UL}), the upper right vertex (C_{UR}), the lower left vertex (C_{BL}) and the lower right vertex (C_{BR}), a scalar value at some position $p = (p_x, p_y)$ within the

[1] Note that the visualization grid is also used in combination with other visualization methods, but since the glyph visualization is the first technique in which the grid is used, it is introduced in this chapter.

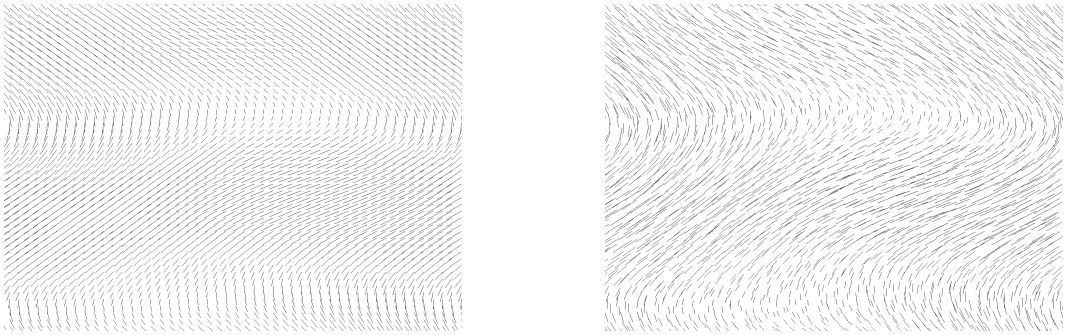
cell is bilinear interpolated according to:

$$\begin{aligned} s = & C_{UL} * (1 - p_x) * (1 - p_y) \\ & + C_{UR} * p_x * (1 - p_y) \\ & + C_{BL} * (1 - p_x) * p_y \\ & + C_{BR} * p_x * p_y. \end{aligned}$$

Note that the the position p is normalized within the cell, i.e. $p_x, p_y \in [0, 1]$. Vectors are interpolated by bilinear interpolation of their components. Since it is unlikely to find inflection points between the sampled points of our vector fields this is sufficient, as long as we use a grid with a reasonable high sampling density.

Jitter

When glyphs are shown on an uniform grid, beating artifacts might appear. These patterns can be avoided by adding some random shifts in the sampling grid. This is done by translating each vertex of the visualization grid with some random values. These absolute values of these translations in both the x and y direction is always smaller than the size of cell, to ensure that the topological ordering of the grid stays the same. The translation in the x direction is determined independent of the translation in the y direction. To determine the translation in some direction a value is sampled from a uniform distribution, with the previous indicated range, the resulting offset is scaled with the jitter factor. This allows us to control how much the grid is jittered. If the jitter factor is zero, the jittered grid reduces to an uniform grid. Figure 3.1 illustrates the effect of adding jitter, the visualization that uses a grid without jitter in figure 3.1a contains beating artifacts, while figure 3.1b does not.



(a) *No jitter* **(b)** *Jitter factor = 0.4.*

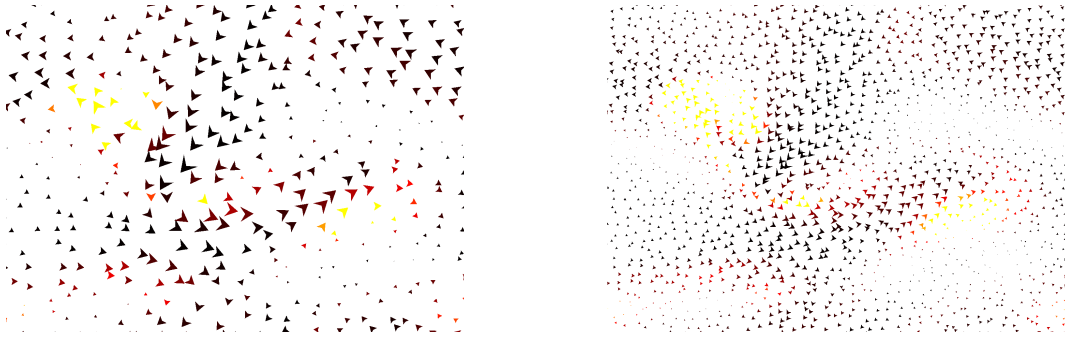
Figure 3.1: *The visualization of a vector field using glyphs (a) without and (b) with jittering.*

3.1.2 Scaling

Glyphs are scaled to show the magnitude of the vector. In our application the size of the glyph is scaled relative to the size of its containing cell. This is done to prevent glyphs from becoming too large which causes cluttering, another advantage is that glyphs will take up more space when the cell size increases, making optimal use of the available space. To ensure the scaling of the glyphs is linear glyphs have no minimum size. This has as disadvantage that glyphs of vectors with a very small magnitude might not be visible, but makes for a more naturally ordered magnitude. The scaling can also be tweaked using a scale-factor which can be set by the user.

3.2 Types of Glyps

This section introduces the different glyphs we support.



(a) *Airplane glyphs on a 20x20 grid.*

(b) *Airplane glyphs on a 40x40 grid.*

Figure 3.2: *In this figure the same simulation is shown twice using airplane glyphs on (a) a* 20×20 *and (b) a* 40×40 *grid.*

3.2.1 Hedgehogs

The simplest form of glyphs are hedgehogs. Hedgehogs are lines which are placed in the direction of the vector and stretched relative to the magnitude of the vector they represent. The glyph can be colored in two ways, first we can let the color of the glyph correspond with the vector magnitude. This way the magnitude is conveyed to the user by both the length and the color of the glyph which can help inverse mapping. Alternatively the glyph can be colored according to some scalar value. This increases the information density of the visualization, but might reduce the clarity of the visualization.

Hedgehogs have as an advantage that the glyphs take up relatively little space. A major drawback of hedgehogs is that it is not possible to distinguish two vectors going in opposite directions, as they are be represented by the same line.

3.2.2 Triangles

The triangle glyph is one of the simplest glyphs that can give directional information. It is constructed by placing the middle of one side of the triangle orthogonal on the vector. The two others sides are stretched in the direction of the vector relative to the vectors magnitude. Thus, the triangle will appear longer as the magnitude of the vector increases. This also increase the area of the triangle, which makes the glyphs representing vectors with a greater magnitude more prominent.

Although triangles properly convey the direction in most cases, a problem arises when the magnitude scales the triangles such that the extended sides become as long as the base side. When this happens the triangle becomes equilateral and can no longer be mapped to a single direction. This could be solved by curving the two extended sides inwards or by making the triangle into an arrow.

The triangle glyph is implemented such that the base side has a fixed length. This has as side-effect that vectors with a small magnitude might appear as lines orthogonal to the vector field. This can confuse users not aware of this effect, who therefore might interpret the triangle as a hedgehog. This increases the learning curve of the visualization. A solution to this problem would be to scale the base of the triangle with the magnitude of the vector.

3.2.3 Airplanes

In the previous section two flaws of the triangle where pointed out that could cause possible difficulty with the inverse mapping process. These flaws are fixed by the airplane glyph discussed in this section. The airplane glyph is a 3D glyph that is defined as two mirrored triangles joined at one side forming an airplane like structure. Depending on the vector magnitude the sides joined together will be pulled upwards, while the free vertex of both triangles will be pulled backwards. This has as effect that the airplane will become longer and pointier as the magnitude increases. Therefore the larger the magnitude becomes, the more prominent the direction of the glyph will be. This has two positive effects. Increasing magnitudes will

have a clear natural order and as the magnitude increases the direction of the glyph becomes better visible. Since one can argue that the direction of a vector is of more importance when its magnitude increases this can be seen as a positive effect.

Since the airplane is a 3D glyph it is possible to use shading with the glyph visualization. This means that in the case of overlapping glyphs, single glyphs can be distinguished more easily due to the visual clues shading offers.

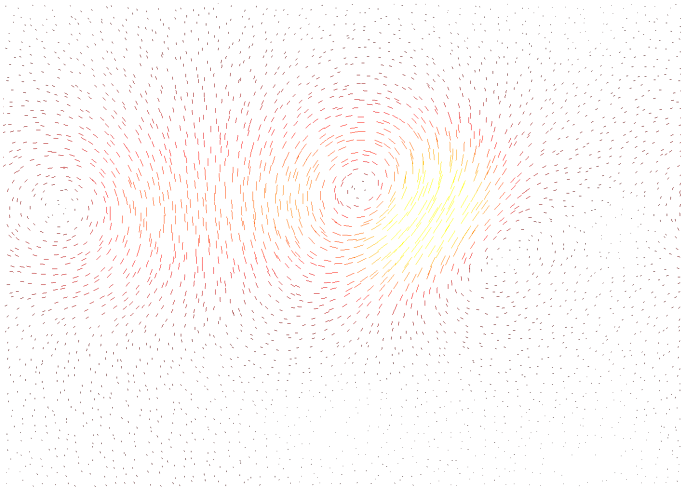
The airplane glyph has a drawback concerning its scaling. When an airplane glyph is scaled up based on some magnitude, its wings are pushed down and its base is pulled up. Because the perspective in which the glyphs are shown (top-down) is not orthogonal to the glyphs surface, the 'wings' of the glyph become skewed, meaning that the visible surface of a glyph is smaller than its real surface. This effects increases as the glyph is scaled up, meaning that its relative surface becomes smaller as its corresponding magnitude increases which means the ordering of the magnitude becomes non-linear.

3.2.4 Results

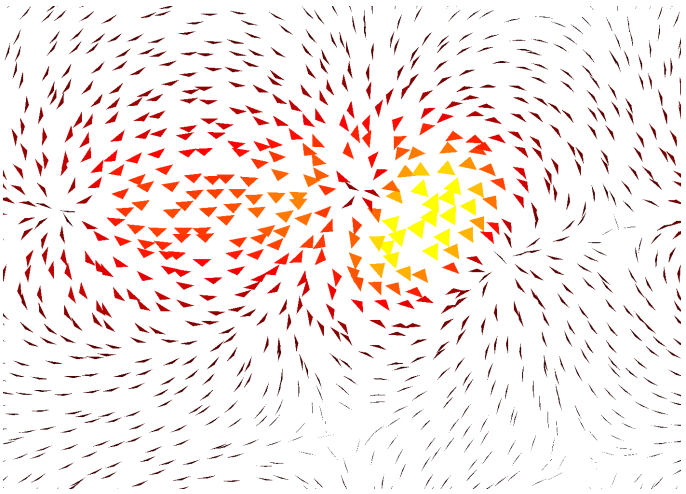
This section presents and discusses three visualizations of the same vector field with different glyphs.

Observing figure 3.3 we can make the following observations. Firstly the hedgehog glyphs in figure 3.3a show the general flow of the vector field. A disadvantage of this visualization is that it is hard to see the size of an individual hedgehog, making it harder to use its magnitude for the inverse mapping to the fluid velocity. Furthermore is it not possible to distinguish sources and sinks since the glyphs do not have a clear origin and destination. Contrary to the hedgehogs the triangle glyphs illustrate the direction of the vector field. We can also see see the magnitude of the vector represented by the glyphs. The disadvantages of these glyphs we mentioned earlier are also visible: when the magnitude of the glyph is near zero the triangles appear as lines and when its magnitude is around its max the triangle becomes nearly equilateral making it harder to see the direction in which they are pointing. Finally the airplane glyphs indicate both direction and magnitude, but are put a greater emphasis on the direction of glyphs with a larger magnitude. The discussed disadvantages are also illustrated by figure 3.3c, as glyphs representing vectors with a large magnitude are some strange 'v' that is hard to observe.

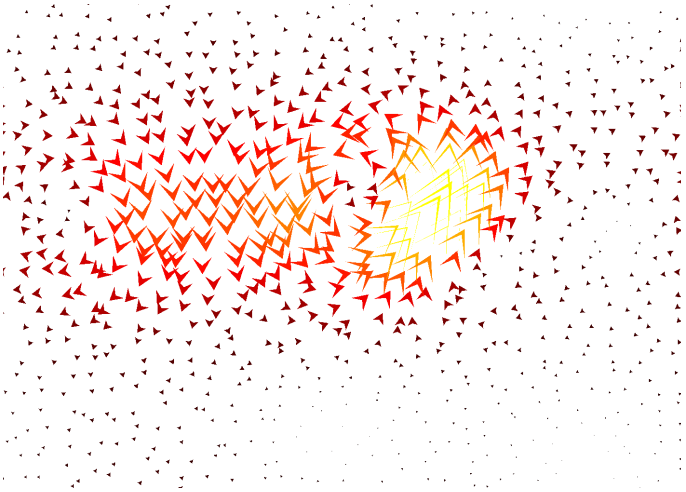
In general we conclude that although glyphs provide a nice way to visualize vector data this method also has many problems. The glyphs require space to be drawn, can occlude each other, sampling artifacts can appear, and directions are sometimes hard to decode. Some of these problems are partly solved by adding jitter to the grid and by using different types of glyphs, but most of these problems cannot be solved completely.



(a) *Hedgehogs on a 60x60 grid.*



(b) *Triangles on a 30x30 grid.*



(c) *Airplanes on a 30x30 grid.*

Figure 3.3: *Three visualizations of the same vector field,* **v***, (a) hedgehogs (b) triangles, and (c) airplane glyphs.*

Chapter 4

Gradients

In the previous chapters we covered the various datasets in the simulation; the fluid density, fluid velocity, and the force field. In this chapter we discuss how two vector fields are created using the existing data: the fluid density gradient and the fluid velocity gradient. Gradients are multi-variable generalization of derivatives and are a function of change on the associated dataset. These vector fields indicate the direction of greatest change and the amount of change in that direction. Since the gradients are not directly represented, they have to inferred from the existing data.

4.1 Calculating Gradients

The fluid density for each vertex of the visualization grid is computed with the process discussed in chapter 3.The gradient of a point on the grid is calculated by looking at the difference between the values of vertices of the cell containing that point and linear interpolating this difference based on the position of the point in the cell. Similar to the interpolation of the vector data, the gradient vector can be interpolated by interpolating the x and y components independently, thus:

$$\Delta \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}.$$

Given the scalar values at the four cell vertices: UL for the upper left, UR for the upper right, BL for the bottom left, and BR for the bottom right vertex the gradient of point p is calculated according to:

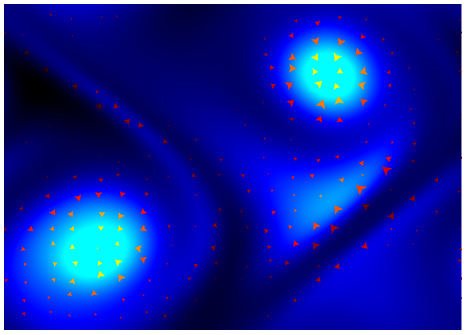
$$\Delta x = (1 - p_y) * \frac{BR - BL}{C_w} + p_y * \frac{UR - UL}{C_w}$$
$$\Delta y = (1 - p_x) * \frac{UL - BL}{C_h} + p_y * \frac{UR - BR}{C_w}$$

This method can be used for both the gradient of the fluid density as well as for the gradient of the fluid velocity magnitude.

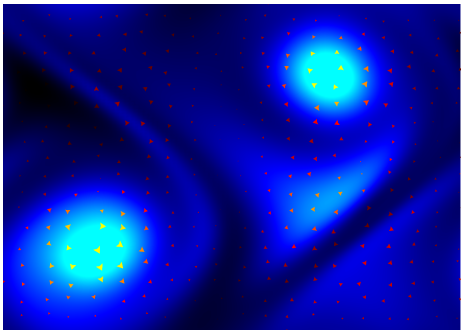
4.2 Visualization of Gradients

Since the gradients are vector data, containing both direction and magnitude, they are best visualized using the glyphs presented in chapter 3. Figure 4.1 shows the gradient vector field visualized with glyphs super imposed on the scalar visualization of the fluid density. Figure 4.1a clearly shows that the gradient points in the direction of the highest rate of change.

The fluid velocity gradient displayed in figure 4.1b appears to be pointing in the opposite direction of the fluid density gradient. Thus fluid moves away from locations which have a relative high density. This effect is even better noticeable when we zoom in on an area with high fluid density, see figure 4.2.

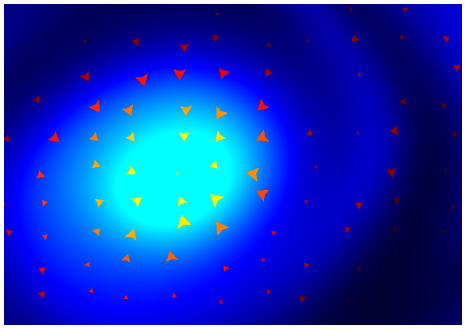


(a) *The fluid density with superimposed airplane glyphs showing the fluid density gradient.*

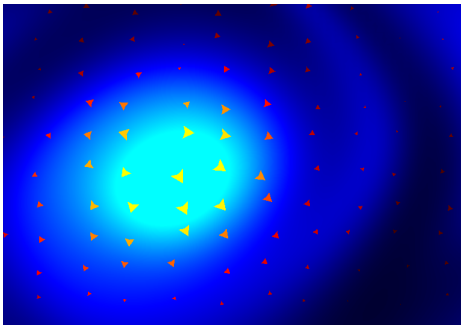


(b) *The fluid density with superimposed airplane glyphs showing the fluid velocity gradient.*

Figure 4.1: *The difference between the fluid velocity gradient and fluid density gradient. Both visualizations have the fluid density as background using the cold colormap and use the heat colormap for the glyphs. Both visualization use airplane glyphs on a* 20×20 *grid using a jitter-factor of 0.2.*



(a) *Fluid density gradient shown in figure 4.1a, zoomed in on an area with max density.*



(b) *Fluid density gradient shown in figure 4.1b, zoomed in on an area with max density.*

Figure 4.2: *A zoomed in version of figure 4.1 showing one of the areas with the highest density.*

Chapter 5

Streamlines

A streamline is the curved path of an imaginary particle that is released at some start location, a seed point, in a stationary vector field[4]. Where glyphs give a general overview of the flow within the vector field, a streamline visualize the localized aspect. Multiple streamlines can of course be used to generate a more general overview of the flow within the vector field. A disadvantage of streamlines compared to glyphs is that they have no direction.

Section 5.1 discuss the computation of streamlines, in section 5.2 the considerations that guided our design of streamlines are presented. And finally section 5.3 shows and discusses our results.

5.1 Method

A streamline can be described as the integral of the vector field $\mathbf{F}(t)$ over some interval T and starting from the location p_0:

$$\mathtt{S} = \{p(\tau), \tau \in [0, \mathrm{T}]\} \quad p(\tau) = \int_{t=0}^{\tau} \mathbf{F}(p)\,\mathrm{d}t \qquad \text{where } p(0) = p_0 \tag{5.1}$$

In this equation t represents integration time, as the vectorfield we consider is time independent, i.e. it is the state of the simulation at a certain point in time. To compute a streamline equation (5.1) needs to be solved. This can be done numerically with Euler intergration:

$$\int_{t=0}^{\tau} \mathbf{F}(p)\,\mathrm{d}t = \sum_{i=0}^{\frac{\tau}{\Delta t}} \mathbf{F}(p_i) \cdot \Delta t \qquad \text{where } p_i = p_{i-1} + \mathbf{F}(i-1) \cdot \Delta t. \tag{5.2}$$

A disadvantage of this integration method is its high integration method. As a consequence the farther away the vertex of the streamline is of the seedpoint the more its position deviatees form the position the imaginary particle would have had if it had traversed the vector field. This could be solved by using an integration method that has a lower error.

Since computing the streamlines is already computationally quite expensive, we chose to use the Euler method, and allow the user to improve the accuracy of the integration by changing t, instead of using an integration method with a lower integration error and higher compotational complexity. To avoid large edges in the streamline we multiplied Δt with the normalized vector [4].

Our integration terminates if one of these conditions is true:

- $t >$ T, i.e. if we have run through our entire time interval.
- The magnitude of p_i is smaller than some lower limit.
- The total length of the streamline is greater than some upper limit.
- p_i falls outside of the computational domain of the simulation.

5.2 Design

The visualization of the vector field with streamlines depends on the placement of the seed points, discussed in section 5.2.1 and the parameterization of the streamline computation, discussed in section 5.2.2.

5.2.1 Seed Points

We have implemented two mechanisms of placing seed points, that can be combined. We allow the user to place the seed points manually. This allows use to use the insight of the user to gain coverage, uniformity and continuity. A disadvantage of placing seed points manually is that it is a lot of work to place seed points like this. Therefore we also offer the probability of placing a grid of seed points over the complete domain. The advantage is that it is fast. Furthermore a fine mazed grid, the grid dimensions can be set by the user, allows for good coverage. However uniformity is definitely not ensured in this way, however this can be solved by letter the user manually add seed points to the grid of seed points near areas where the density of the streamlines is low.

5.2.2 Visualization Parameterization

The user can parameterize the visualization with a number of variables. The `time step`, `timeStep`, and `maximum time`, $\texttt{timeStep}_{\max}$, directly map to t and T. The length of the edges of the streamline, and the upper limit of the length of the streamline are controlled by `edgeLength` and $\texttt{streamLineLength}_{\max}$, respectively. Both of these are expressed in the width, or equivalently since our simulation cells are squares, in the height, of the simulation cells. This gives the user more insight in the length of the (elements of the) streamlines, than arbitrary numbers would have been. Lastly the user can also set the minimum magnitude, `magnitudeMin`, to control the termination of the integration of the streamline.

Both `timeStepMax` and `edgeLengthMax` can be set to ∞, this makes it possible to discount these factors when computing the streamlines. If both of these values are set to ∞ the integration of the streamline only terminates if the magnitude of the vector falls below $\texttt{magnitude}_{\min}$ or if p_i falls outside of the computational domain.

5.3 Results

This section presents several visualizations of the fluid velocity. All visualizations use the rainbow color map with 256 colors, without clamping and with the saturation set to 1.0.

Figure 5.1 shows the visualization of the vector field with streamlines when the seed points are manually placed in areas with a high fluid velocity magnitude. This visualization has poor coverage, a large area of the vector field is not visualized at all. In the areas where streamlines are present the visualization is reasonable uniform, although the streamlines deviate quite a lot in the lower left corner. The continuity of this visualization if pretty good, near the center a few relatively long streamlines are present, but these are not so short as to make them hard to interpret. Figure 5.2 illustrates the influence of the minimum magnitude. Clearly figure 5.2a is useless, and figures 5.2b and 5.2d are hardly useful. Comparing figure 5.2e with figure 5.2f we see that the area around the points of zero magnitude has significantly decreased.

In figure 5.2 the coverage is good for $\texttt{magnitude}_{\min} < 0.0001$. For these values of $\texttt{magnitude}_{\min}$ uniformity is better than it was in figure 5.1, but especially near the borders of the computational domain the density of the streamlines is lower than near the region of high fluid density velocity. All streamlines are long enough to follow, continuity is good for $\texttt{magnitude}_{\min} < 0.0001$.

Figure 5.3 shows the influence of the time step on the visualization. Clearly increasing `timeStep` results in a visualization with increasingly shorter streamlines. In figure 5.3d the streamlines have become short enough that the resulting visualization is reminiscent of the visualization of this vector field with glyphs. Although the continuity leaves much to be desired for $\texttt{timeStep} = 50$, this visualization has good uniformity and coverage. The visualizations with lower values for `timeStep` have better continuity, but worse coverage and uniformity. Based on figure 5.3 we observe that as `timeStep` increases continuity decreases.

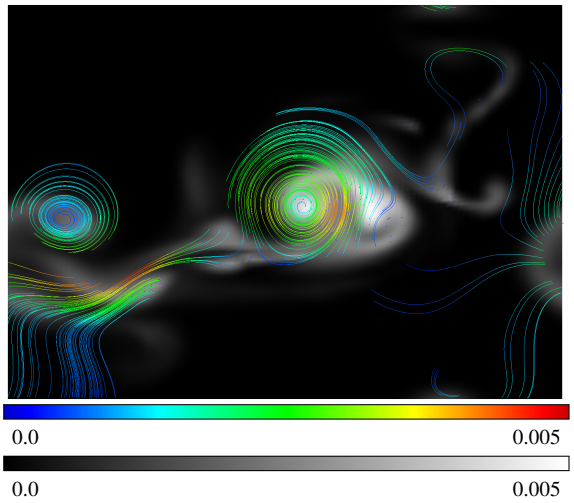
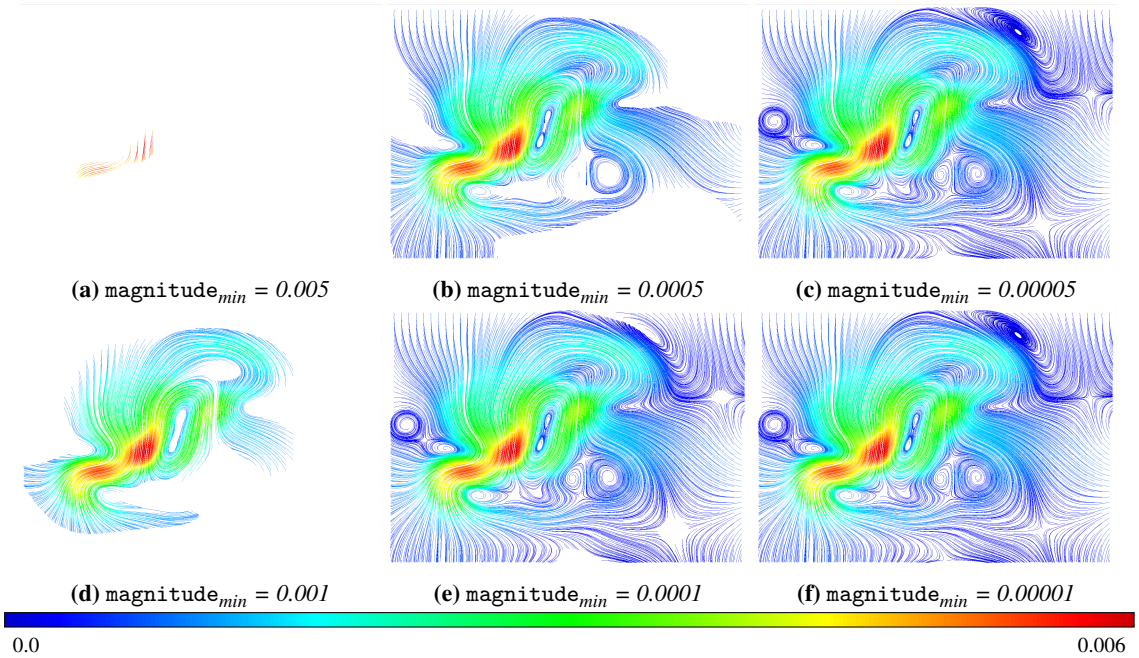


Figure 5.1: *Manually seeded streamlines superimposed on scalar visualization of the fluid density with a gray scale color map that shows the fluid velocity magnitude. Seed points were placed in areas of high fluid velocity magnitude.* `timeStep` = *1.0,* $\texttt{timeStep}_{max}$ = *100.0,* `edgeLength` = *0.33,* $\texttt{streamLineLength}_{max} = \infty$.



(a) $\texttt{magnitude}_{min}$ = *0.005* **(b)** $\texttt{magnitude}_{min}$ = *0.0005* **(c)** $\texttt{magnitude}_{min}$ = *0.00005*

(d) $\texttt{magnitude}_{min}$ = *0.001* **(e)** $\texttt{magnitude}_{min}$ = *0.0001* **(f)** $\texttt{magnitude}_{min}$ = *0.00001*

Figure 5.2: *Streamlines, seeded on an uniform* 50×50 *grid of seed points with varying minimum magnitude.* `timeStep` = *1.0,* $\texttt{timeStep}_{max}$ = *100.0,* `edgeLength` = *0.33,* $\texttt{streamLineLength}_{max} = \infty$.

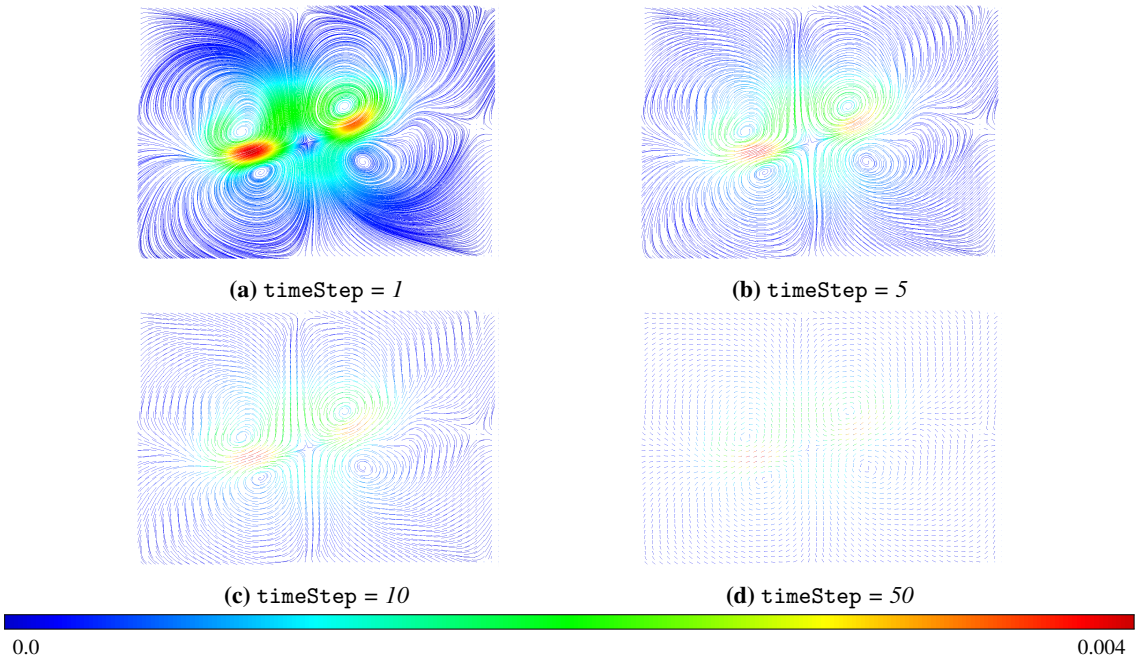
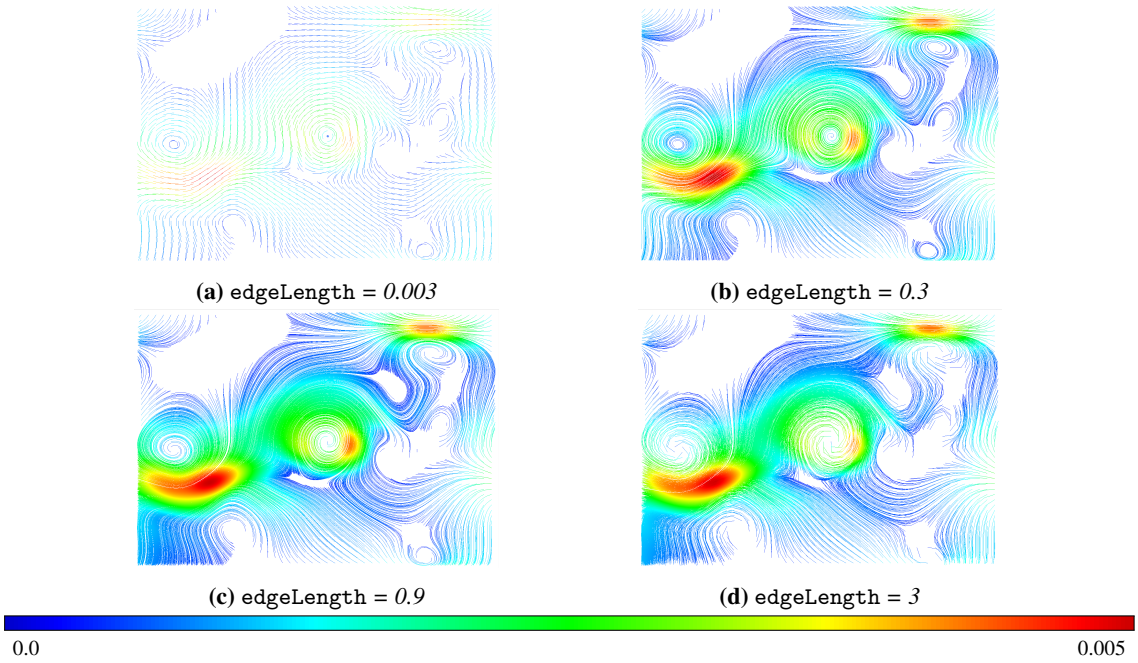


Figure 5.3: *Streamlines, seeded on an uniform* 50×50 *grid of seed points with varying time steps. Seed points are shown as blue dots.* $\texttt{timeStep}_{max}$ *= 100.0,* `edgeLength` *= 0.33,* $\texttt{streamLineLength}_{max} = \infty$*,* $\texttt{magnitude}_{min}$ *= 0.00001.*

If we were to increase the maximum time the streamlines would get longer. Eventually the integration of streamlines would only be terminated because the current magnitude falls below the threshold for that value. Figure 5.4 illustrates the effect of varying edge lengths. None of the visualizations in this figure are uniform or have globally good coverage. Figure 5.4a has good local coverage and uniformity and coverage. But quite low continuity, comparing this figure with the visualizations of the same vector field with a different value of `edgeLength` we see that the integration of the streamlines in figure 5.4a has terminated because $t > \mathrm{T}$. The differences in density between figures 5.4b and 5.4c illustrate show that length of the streamlines for $\texttt{edgeLength} = 0.03$ is terminated for that same condition. In figure 5.4d we observe that a too high edge length results in very angular streamlines. Clearly `edgeLength` is of great influence of the continuity of the streamlines, which in term influences the coverage.

The bottom edge of the lower left corner of this figure shows that we simply discard the next vertex on the stream line if it went outside of the computational domain, instead of following the vector to the edge of the computational domain. The computations for this are quite trivial, but given that normally `edgeLength` is quite small not extremely relevant.

Decreasing the maximum length of a streamline would do exactly what the name of its parameter implies, as soon as the maximum length is low enough that the integration of the streamlines is not terminated because of a too low magnitude or because $t < \mathrm{T}$.



(a) `edgeLength` = *0.003* **(b)** `edgeLength` = *0.3*

(c) `edgeLength` = *0.9* **(d)** `edgeLength` = *3*

Figure 5.4: *Streamlines, seeded on an uniform* 50×50 *grid of seed points with varying edge lengths.* `timeStep` = *1.0,* $\texttt{timeStep}_{max}$ = *100.0,* $\texttt{streamLineLength}_{max} = \infty$, $\texttt{magnitude}_{min}$ = *0.0004.*

Chapter 6

Slices

In the previous chapters we have shown visualization techniques that only showed one time-frame of the simulation at once. With slices we have implemented a technique that extend these visualization by stacking multiple time-frames of the simulation on top of each other. This creates a 3D simulation in which the x and y axis correspond with the grid of the simulation and the z axis corresponds with the time. This way a 3D volume is created which shows the visualization over time.

6.1 Stacking Slices

A slice of the visualization can be considered as one frame of the visualization at a certain time-step in the simulation. Slices can, just as the 2D visualization, consist of scalar visualization using color maps, glyphs, streamlines, or any combination of the previous. By stacking the slices on top of each other, a volume is formed showing a 3D visualization of (historical) simulation data.

6.1.1 Combining Slices

In the section above we assumed that every time-step a new frame was pushed added to the stack of slices. However, this might result in a rapidly changing volume which can be hard to interpreted. Furthermore this limits the time-frame that is captured, since the number of slices that can be shown is limited. This can be solved by considering multiple times-steps per each slice. The simplest method to achieve this is by only displaying one frame of every n time-steps. However, this way a lot of information is thrown out. Another approach is to combine the n frames by taking their average. This has as advantage that every frame is somewhat represented in the visualization, but it might blend out opposite movements. Note that the data are averaged before the visualization is computed. Thus we do not combine the average of several glyphs, but compute the average of a number of states of the simulation, we use that to compute the glyphs.

6.2 Infrastructure

To support the slices a slice-engine is build for each visualization technique. This slice-engine holds a size-limit queue in which slices can be stored. A size limited queue has FIFO buffer behavior.

- New slices are added to the top of the queue.
- When a new slices is added, all slices in the queue will be moved down one slot.
- Once the queue has reached its capacity, the oldest slice will be dropped when a new slice is added ensuring the queue will not be over capacity.

The reason a size limited queue is used is performance. Slice visualization is computationally heavy and storing too much slices can slow the visualization to a halt. In our application the maximum number of

slices is hundred, with the default set to twenty. For further optimization we store the content of the buffers that send to the GPU. A disadvantage of this approach is that if we change the parameterization of the computation of the visualization, for example we go from triangle glyphs to airplane glyphs, the old data have to be deleted.

6.3 Slice Visualization

Slices are visualized by associating each slice with a z-value and drawing the slice in the xy-plane defined by the given z. In the visualization the slices are ordered such that the newest slices are drawn at the top (and thus have a high z-value), while the older slices are at the bottom.

6.3.1 Alpha Blending

To enable the user to look inside the 3D volume alpha-blending is used. By lowering the alpha-channel the slices become somewhat transparent allowing the user to see multiple slices 'through' each other. The amount of alpha-blending is regulated in two ways.

A global alpha value can be set by the user which controls the alpha value of each element in the visualization. Next this global alpha value is adapted for each data-point in the simulation depending on the scalar value or vector magnitude of the visualized data. This is done by scaling the global alpha value linearly with the scalar value, normalized by the current range of the scalar data. This means high valued data will also have a high alpha value. This means maximum values will be emphasized while minima will be blended out.

Alpha blending has as disadvantage that visualizations will becomes 'smudged' and are not clearly visible anymore. Furthermore alpha-blending can cause false colors.

6.3.2 3D Viewing

An important part of the slice visualization are the controls which can be used to change the viewing perspective. These controls enable the user to rotate the 3D volume of slices around the three axis, zoom in and zoom out, and move the slices relative to its position. They enable the user to change their viewpoint, to inspect the different sides of the 3D visualization. By manipulating their view-point the user can gain more insight compared to a fixed perspective. Since the controls are a bit hard to use, two often used perspectives are preset. The top down view and the side view.

6.4 Results

In this section various results of visualization using slices are given. Take note that these visualizations are very information dense and are therefore not very well suited to be viewed on paper. In figure 6.1 the slices visualization for scalar visualization using color mapping. Looking at the slices one can see how the area with high fluid density moves trough the x y space. This is most prominent in the side view, but also visible due to the alpha-blending in the top down view.

In figure 6.2 a slices visualization of streamlines is given. Here we can see the streamlines moving in a sort of waving fashion. While the result for the color map were most prominent in the side view, the streamlines benefit most from the top down view.

(a) *Slices showing fluid density using the divergent color map from a side view.*

(b) *Slices showing fluid density using the divergent color map from top down view.*

Figure 6.1: *Slices visualization combining 20 frames of a scalar color map visualization into a 3D visualization.*

(a) *Slices showing the fluid density visualized using streamlines from a side view.*

(b) *Slices showing the fluid density visualized using streamlines from a top down view.*

Figure 6.2: *Slices visualization combining 20 frames of streamlines visualization into a 3D visualization.*

Chapter 7

Stream Surfaces

Stream surfaces are basically a bunch of connected streamlines that all have their seed point on some seed curve. The flow of the vector field is always tangent to the surface of stream surface. As our simulation has only two dimensional vector fields we use time as the third dimension. Thus our stream surfaces show the variation of a two dimensional vector field as a function of time.

Section 7.1 explains how we compute the three dimensional streamlines, and how we go from a streamlines that are densely seeded on a curve to a surface. Section 7.2 is concerned with the visualization of the thus defined surface. In section 7.3 a number of visualizations of a three dimensional vector field are presented and discussed.

7.1 Method

This section explains how one can define a stream surface given a seed curve Q, with n vertices, $q_0, \cdots, q_n$ and T two-dimensional vector fields, $\mathbf{F}_0(), \cdots, \mathbf{F}_T()$, that represent the different states of some vector field as a function of time.

In section 7.1.1 we explain how we compute a three dimensional streamline in a number of consecutive two dimensional datasets. Section 7.1.2 explain how we compute the seed points given the seed curve Q. Section 7.1.3 discusses how we define a stream surface from a set of streamlines.

7.1.1 Time as the Third Dimension

We start the streamline at seed point p_0 in the two-dimensional vector field $\mathbf{F}_T()$, i.e. the oldest vector field that we have. We then compute the next vertex of the two-dimensional streamline in the vector field $\mathbf{F}_T()$ with seed point p_0. We refer to this two-dimensional position as p_1. This position is used as the seed point of a two-dimensional streamline of one edge in the vector field $\mathbf{F}_{T-1}()$. The other vertex of that edge is referred to as p_2, and is used as the seed point in $\mathbf{F}_{T-2}()$. This process is repeated until we have reached $\mathbf{F}_0()$.

The points $p_0, \cdots, p_T$ define the three dimensional stream line. The z-coordinate of these points is computed in such a way that the points are spread uniformly in z over some predefined range for z.

The streamline is thus computed according to the process described in chapter 5, with one change: there is no lower bound for the magnitude, i.e. $\texttt{magnitude}_{\text{min}} = \infty$.

7.1.2 From a Seed Curve to Seed Points

The seed curve is a polyline, each of the line segments of this polyline is split into `resolution` segments. The higher `resolution` is the better the final stream surface is.

7.1.3 Building the Surface

The final step of defining a stream surface is the determination of a surface from a set of streamlines, $\mathtt{S}_0$, $\mathtt{S}_N$. The line segments of the stream lines are have all approximately the same length, as the distance between p_i and p_{i+1} is equal for every segment in every stream line and the difference in z-coordinates of consecutive stream line vertices is also equal. We then define the stream surface by connecting vertex p_i of stream line $\mathtt{S}_j$ with p_i of stream line $\mathtt{S}_{j+1}$, for $j < N$. If stream line $\mathtt{S}_j$ has a vertex p_i, but $\mathtt{S}_{j+1}$ is shorter and does not have a p_i, we connect p_i of $\mathtt{S}_j$ to the last vertex of $\mathtt{S}_{j+1}$.

The resulting connections define a collection of quads that could define a surface. However the thus defined surface would happily move though some obstacle that is encountered by the flow of the vector field. To avoid this we do not connect two neighboring vertices of streamlines if the distance between them is greater than $\mathtt{divergence}_{\max}$. We chose to explicitly model the divergence of stream lines in this way as to us, adding more seedpoints at the location of the divergence did not make sense.

7.2 Design

The visualization of the vector field with the stream surface depends on the placement of the seed curve, which is discussed in section 7.2.1 and the parameters discussed in section 7.2.2.

7.2.1 Seed Curve

Finding the best position for the seed curve is far from trivial to do automatically. Instead we ask the user to use their insight and draw the seed curve.

7.2.2 Visualization Parameterization

Several parameters influence the visualization of the seed curve. We have already mentioned the number of seed points that is placed on one line segment of the seed curve, i.e. `resolution`.

The number of states parameter, `#states`, maps directly to the parameter T, mentioned in section 7.1.1. The higher this parameter is the longer the stream lines can become.

The parameter $\mathtt{divergence}_{\max}$ controls the maximum distance that is allowed between two neighboring vertices of two neighboring stream lines, as discussed in section 7.1.3.

The generation of the stream lines is influenced by the parameters discussed in section 5.2.2, we have set these as follows: $\mathtt{timeStep} = 1.0$, $\mathtt{edgeLength} = 0.33$, $\mathtt{streamLineLength}_{\max} = \infty$.

7.3 Results

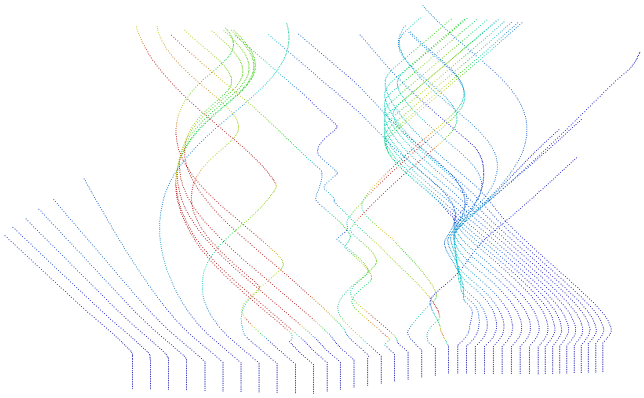
This section presents and discusses several visualizations of the evolution of a two-dimensional vector field as a function of time with stream surfaces. All visualizations in this section will show the fluid velocity, and will be colored according to the fluid velocity magnitude with a color map that is not clamped. The older simulation states are shown near the bottom of the images, more recent states are shown near the top of the images.

We support three different ways of visualizing a stream surface. Firstly one can use the vertices of the streamlines as shown in figure 7.1a. Alternatively one can visualize the streamlines as in figure 7.1b, or the surface defined by these lines as in figure 7.1c.

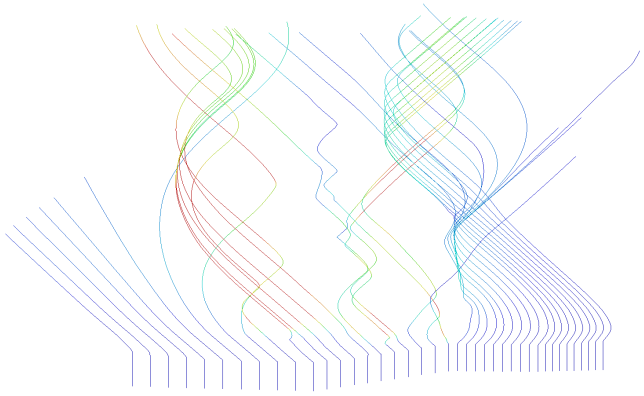
Figure 7.1c clearly illustrates one issues of our stream surfaces, firstly the edges of the computational domain are not reached gracefully. It is possible that this is caused by the fact that our streamlines end at their last vertex before the end of the computational domain, instead of on the edge of the domain.

It should be noted that the resolution of seed lines used for the visualizations in figure 7.1 was chosen quite low, to emphasize the difference between the lines and the vertices.

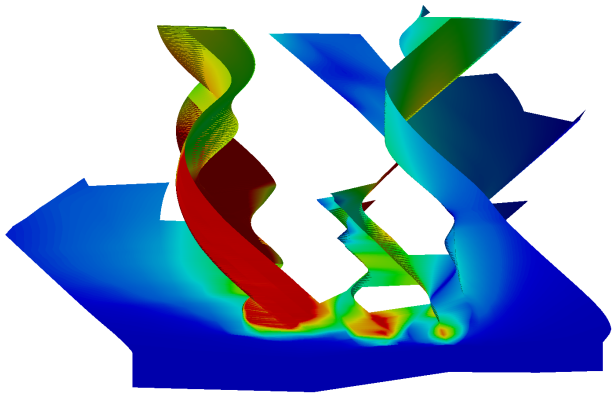
The straight streamlines at the bottom of these figures, i.e. in the oldest simulation states, are caused by lack of user input at the beginning. In that case, all vectors are $\mathbf{0}$, consequently $p_i = p_{i+1}$, which results in a line segment whose vertices only differ in their z-values.



(a) *Vertices*

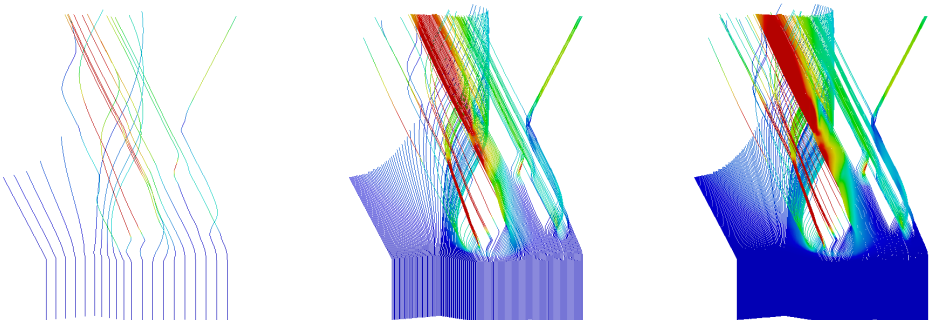


(b) *Stream Lines*



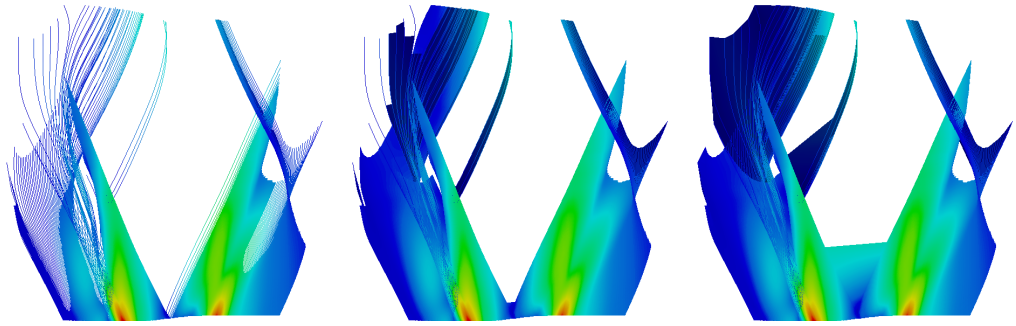
(c) *Stream Surface*

Figure 7.1: *Visualization of the change of a two-dimensional vector field,* $\mathbf{v}$*, as function of time with (a) the vertices of the stream lines, (b) the stream lines and (c) the stream surface.* `resolution` *= 10,* `#states` *= 200,* $\texttt{divergence}_{max} = 200.0$*.*



(a) `resolution` = *5* **(b)** `resolution` = *50* **(c)** `resolution` = *100*

Figure 7.2: *Visualization of the change of a two-dimensional vector field,* **v***, as function of time with different resolutions.* `#states` = *200*



(a) `resolution` = *3* **(b)** `resolution` = *30* **(c)** `resolution` = *200*

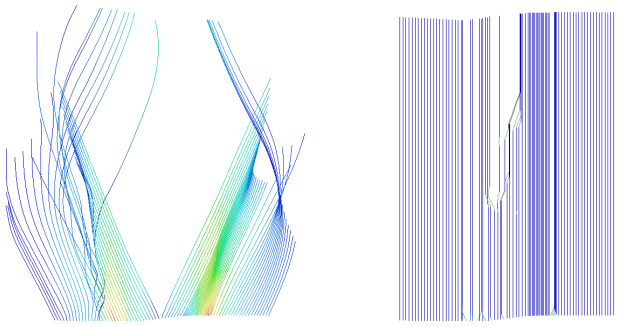
Figure 7.3: *Visualization of the change of a two-dimensional vector field,* **v***, as function of time with different divergence criteria.* `resolution` = *100,* `#states` = *200.*

Figure 7.2 presents three visualizations of the same data set with three different resolutions. This figures clearly illustrates the importance of choosing a resolution that is sensible for the data, in this case a resolution of five is clearly too low, `resolution` = 50, seems sufficient. If resolution is set to 100 the stream lines are so close together that they nearly form a surface.

The influence of the $\texttt{divergence}_{\max}$ is illustrated in figure 7.3. To more clearly show the divergence both the stream lines and the stream surfaces are shown. Since the length of the stream lines are not influenced by $\texttt{divergence}_{\max}$ they are always visible independent of the divergence of the vector field.

The influence of the $\texttt{divergence}_{\max}$ can be seen in the size of the triangle that is placed between the flow that moves to the left and the flow to the right. The higher the divergence criterion the longer the two flows are connected. If $\texttt{divergence}_{\max} = 3$ the stream splits quite early, whereas for $\texttt{divergence}_{\max} = 200$ the streams stay together until $^1/_3$ of the y-axis. The influence of the divergence criterion is also illustrated in the upper left corner of these images.

Figure 7.4 shows the stream lines of two different vector fields, namely the fluid velocity field and the force field. Figure 7.4b clearly illustrates that the added forces decays quite quickly. This visualization also shows that the stream lines are straight lines as long as the vector fields don't change as a function of time at the position of the seed point.



(a) v **(b) f**

Figure 7.4: *Visualization of the change of (a) fluid velocity field and (b) the force field as a function of time.* `resolution` = *100*, `#states` = *200*

Bibliography

[1] David Borland and Russell M Taylor Ii. “Rainbow color map (still) considered harmful”. In: *IEEE computer graphics and applications* 27.2 (2007).

[2] Kenneth Moreland. “Diverging Color Maps for Scientific Visualization”. In: *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*. ISVC ’09. Las Vegas, Nevada: Springer-Verlag, 2009, pp. 92–103. ISBN: 978-3-642-10519-7. DOI: 10.1007/978-3-642-10520-3_9. URL: http://dx.doi.org/10.1007/978-3-642-10520-3_9.

[3] Jos Stam. “A Simple Fluid Solver Based on the FFT”. In: *J. Graph. Tools* 6.2 (Sept. 2002), pp. 43–52. ISSN: 1086-7651. DOI: 10.1080/10867651.2001.10487540. URL: http://dx.doi.org/10.1080/10867651.2001.10487540.

[4] Alexandru C. Telea. *Data visualization: principles and practice*. CRC Press, 2014.