rijksuniversiteit groningen / faculteit wiskunde en natuurwetenschappen / informatica

Scientific Visualization

Fluid Flow Simulation and Visualization

Eduardo Faccin Vernier - s3012875

Groningen, 13th of April 2016

Table of Contents

Introduction

Summarizing in very few words, the assignment consist in, given a real-time fluid flow simulation, implementing several visualization techniques in order to get insight on the fluid movement.

These techniques are applied over scalar and vector fields that describe the phenomena that are taking place.

In this assignment, I chose to use the GLUI User Interface Library, as it is simple to use and has enough resources to guarantee a good user interface. For debugging purposes, the QtCreator IDE was used.

Students were given two variants of the assignment to choose from and I choose to follow branch A.

Step 1: Skeleton Compilation

As I chose to work with the C++ programming language, this step was initially quite easy and no modifications were needed in the makefile. I then decided to refactor the code to a more Model-View approach, but that proved to be harder than expected, as GLUT only receives static methods as arguments to its displaying and reshaping functions, and that caused some trouble on the classes implementation, leading me to use a hybrid approach.

The integration of GLUI to the project was a little problematic at first, but the library proved to be very fit for the project.

Step 2: Color Mapping

In this step we were asked to implement a set of color mapping techniques to be applied to three scalar fields: the fluid density *rho*, the fluid velocity magnitude ||v|| and the force field magnitude ||f||. The first was already given in the skeleton code, the other two had to be computed on the fly.

Initially, the technique used to color the polygons was linear interpolation between the vertices (vertex-based color mapping), but as shown in section 5.2 of the course textbook, it is a flawed method because it allows, under some circumstances, colors outside the colormap to be displayed. A better alternative is the use of texture-based color mapping, as in this technique, the interpolation results are not a product of the vertices colors (which might generate false colors), instead it interpolates the scalar value and for every individual pixel in the polygon and maps it to a 1D texture that works as a lookup table. The results obtained with this methods are

much superior and contain no false colors. The image below compares the two rendering techniques. In both images a 50 by 50 simulation grid is used.

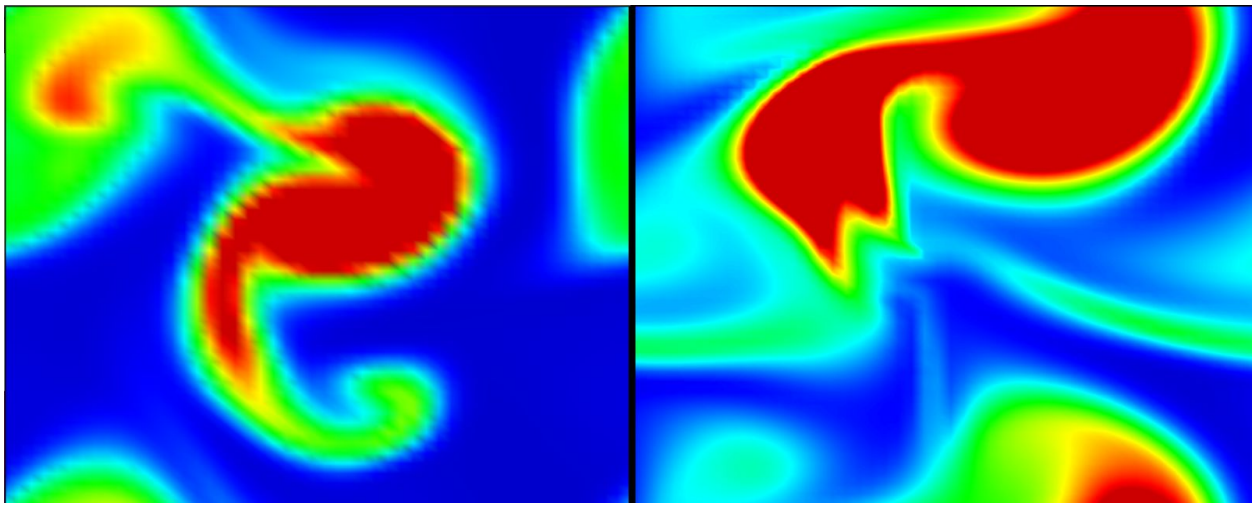

Image 2.1 - Vertex-base color mapping on the left and texture based color mapping on the right for different states of the rho scalar field using the same rainbow colormap.

There are three pre-set colormaps implemented in the program and a fourth one can be parameterized by the user. The image below displays the rainbow colormap, the grayscale colormap, the heat colormap and a custom colormap, which was defined in run time with the following parameters: black in the interval $0 \leq s < 0.2$, white in the interval $0.2 \leq s < 0.4$, black in the interval $0.4 \leq s < 0.6$, white in the interval $0.6 \leq s < 0.8$, black in the interval $0.8 \leq s < 1$ and white for $s \geq 1$, describing a variation of the zebra colormap.

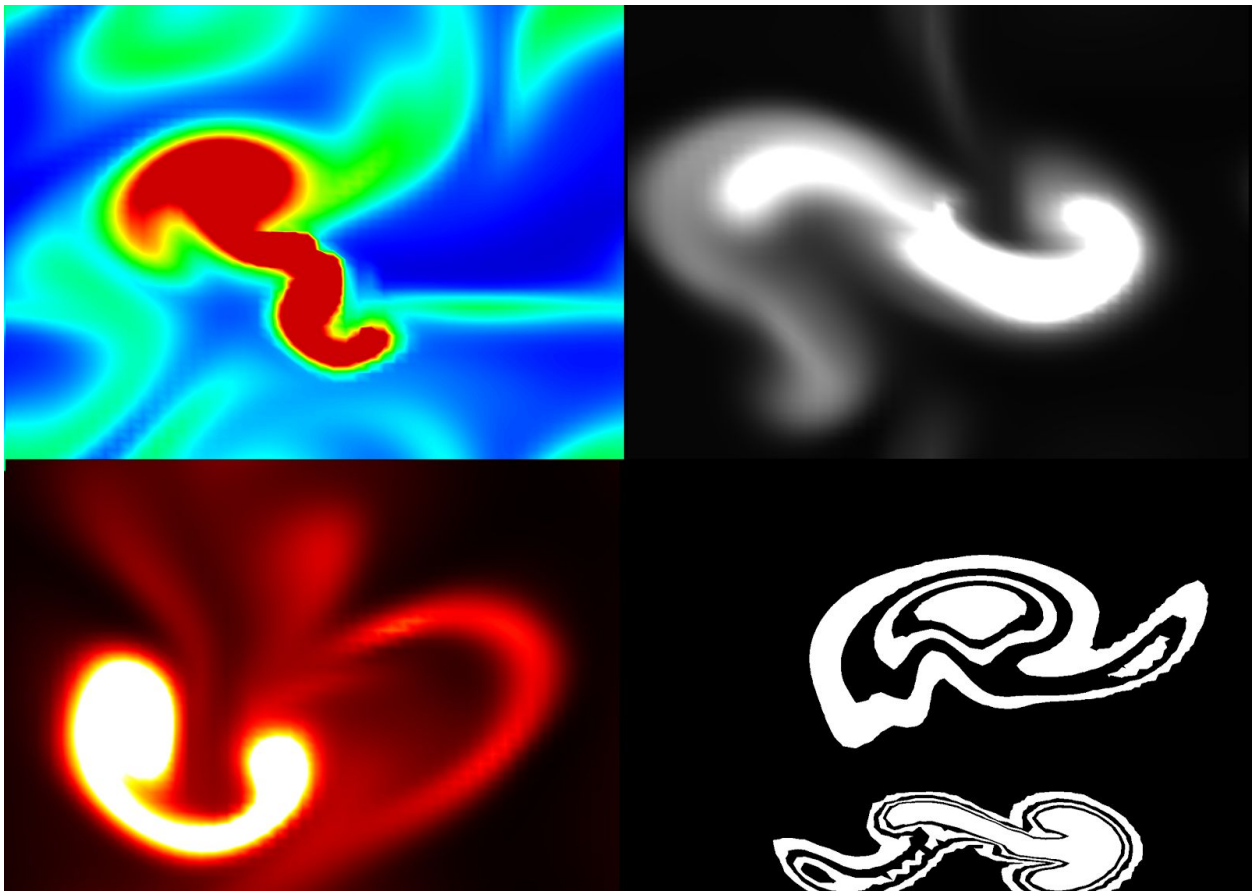

Image 2.3 - Colormaps present in the implementation.

The custom colormap can be parametrized as wished by the user, therefore it can be used to display contouring lines. Even though this method doesn't produce results as good as the ones obtained with algorithms such as Marching Squares (more on that on Section 5a), it helps us easily distinguish which are the points that have a scalar value of interest.

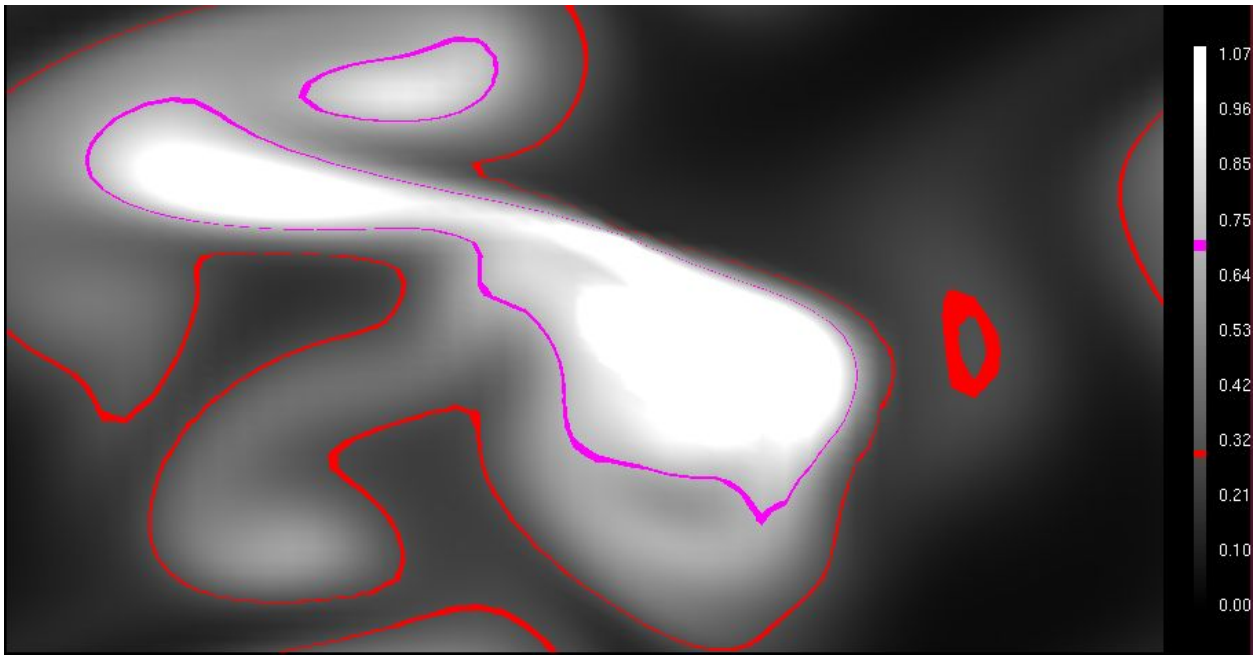

Image 2.2 - Grayscale colormap with s = 0.3 highlighted in red and s = 0.7 highlighted in purple.

The user can also opt for the quantization of a colormap. This technique, also called color banding, is displayed in the following image for colormaps with 256, 16, 8 and 4 colors. Color banding may allow us to get interesting insight on the scalar field we are studying, such as where some values are exceeded, and maybe a clearer understanding of the values distribution, but this dismissal of information is usually not desirable when dealing with continuous datasets.

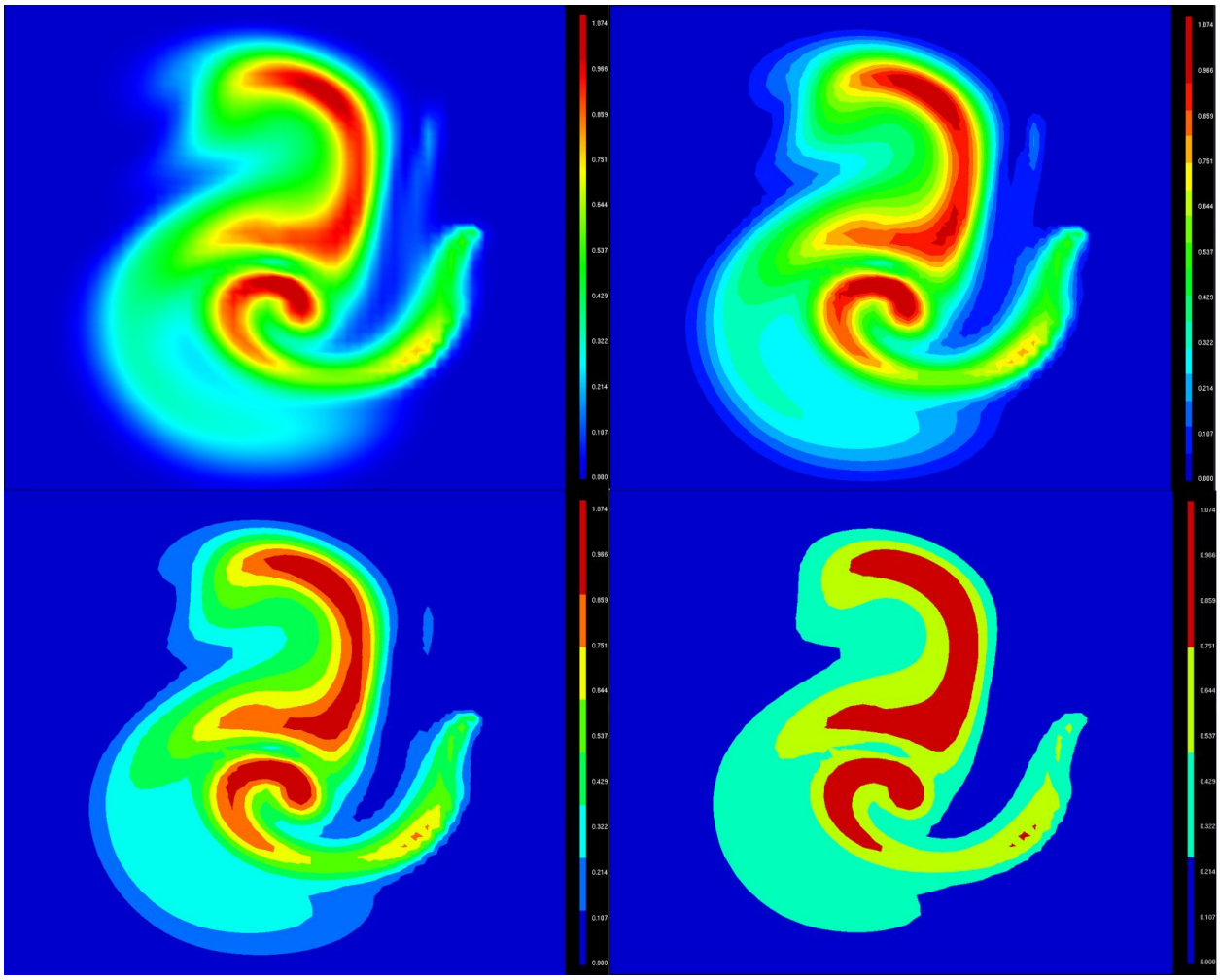

Image 2.4 - Colormaps reduced to 256, 16, 8 and 4 colors displayed on the same density scalar field.

Two mechanisms to apply the colormap on the dataset were implemented: clamping and scaling.

Clamping - given *min* and a *max* values chosen by the user of the application, these values are made to correspond to the entire range of the color legend. This means that the colormap that was by default clamped between (0, 1) is now clamped between (*min*, *max*).

Scaling - the colormap is now made to correspond to the range between the highest and lowest in the dataset.

These operations result in the following results:

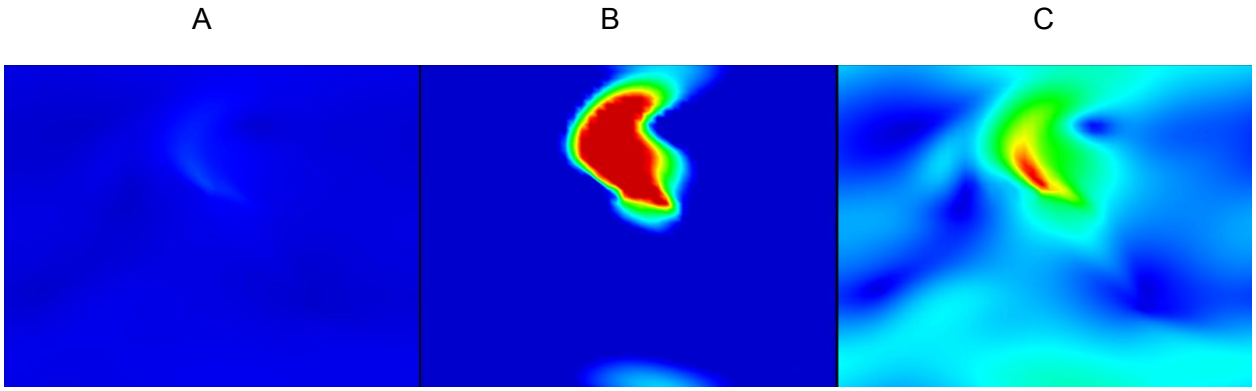

Image 2.5 - Scalar visualization of the force magnitude filter with (A) no mechanism applied, (B) with clamping (0.03, 0.05) and (C) with scaling.

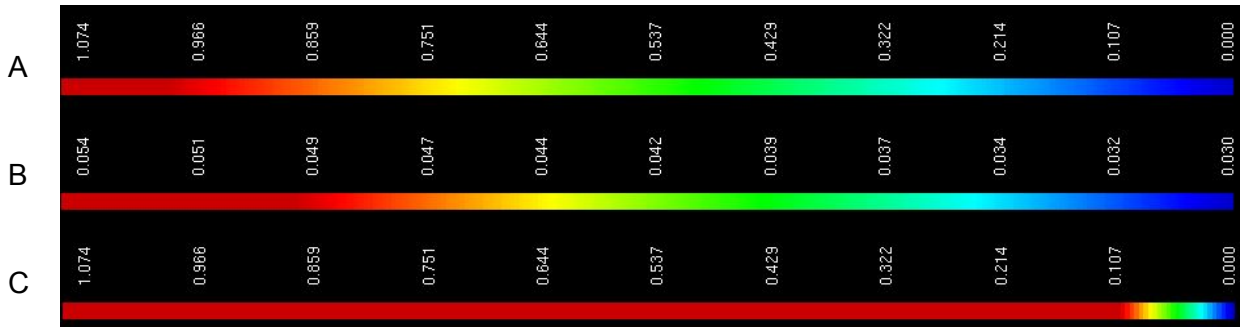

Image 2.6 - Colormaps representing the three mechanisms described above. Notice how in B, the numerical values change, whilst in C the whole colormap shrinks.

Step 3: Glyphs

In this application we are dealing with two vector fields, fluid velocity **v** and force field **f**, and vector glyphs is the simplest technique to visualize this kind of data. In a glyph we can encode vector field values as well as independent scalar field data, by coloring it using an appropriate colormap, for example.

Three models of glyphs were implemented in this application: line, arrow and needle. Their shapes and orientations are used to represent the state of the vector fields **v** or **f**. Their color can be also used to encode information about the magnitude of these vector fields as well as the fluid density on the glyph origin point and vector field direction. It is also possible to scale the glyphs and choose the number of glyphs to be displayed on screen. An attempt to design a 3D glyph that uses shading as a visual cue to more easily separate overlapping glyphs was made, but unfortunately, I wasn't able to get it working properly.

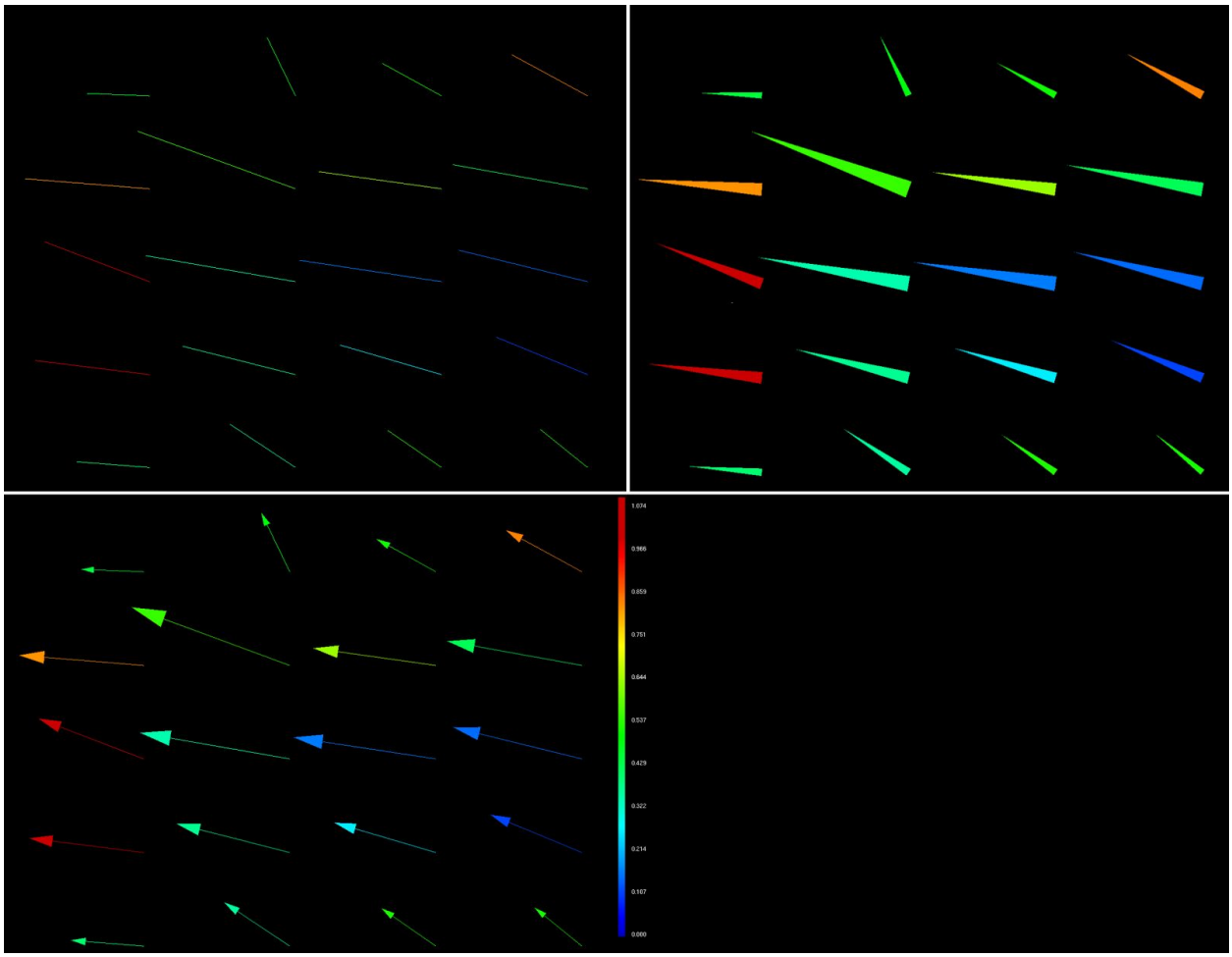

Image 3.1 - Three glyph icons illustrating the same vector field state. They are arranged in a 5x4 regular grid and are scaled according to the vector field magnitude in their origin point. Their color is set by the fluid density in the origin point using the rainbow colormap.

The four following images illustrate the same fluid velocity vector field state. The first uses a 50x50 regular grid, which is the same dimension as the simulation dataset, whilst the other three use randomly jittered positions as origin points.

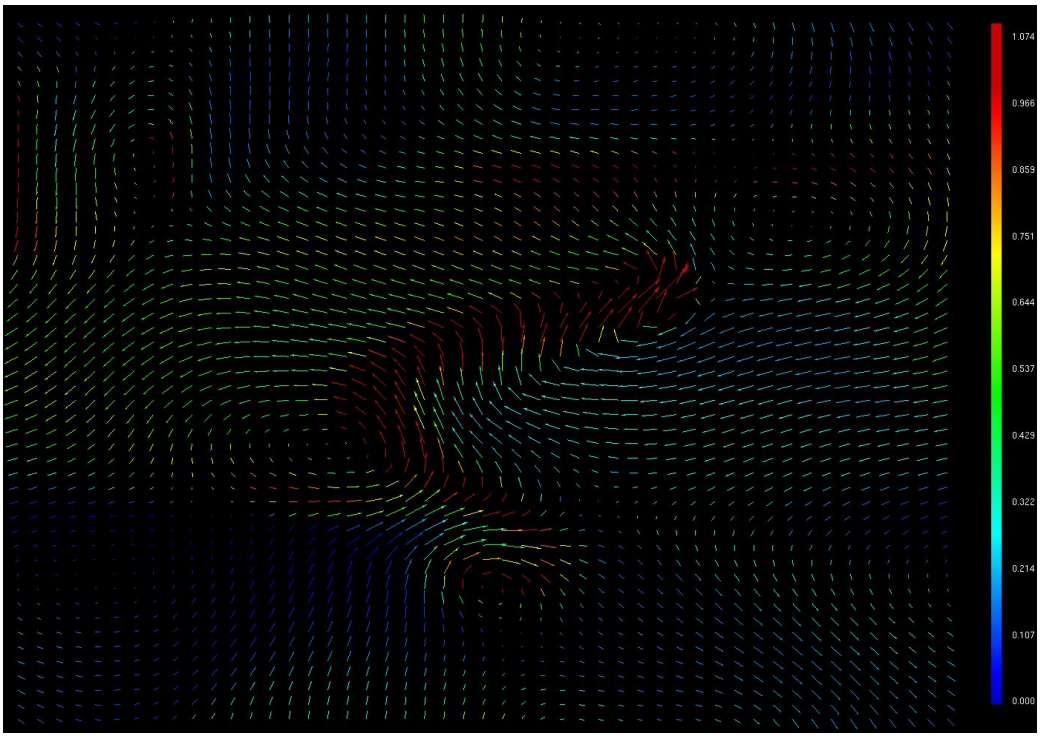

Image 3.2 - 50x50 uniform grid using arrow glyph and rainbow colormap based on the fluid density field.

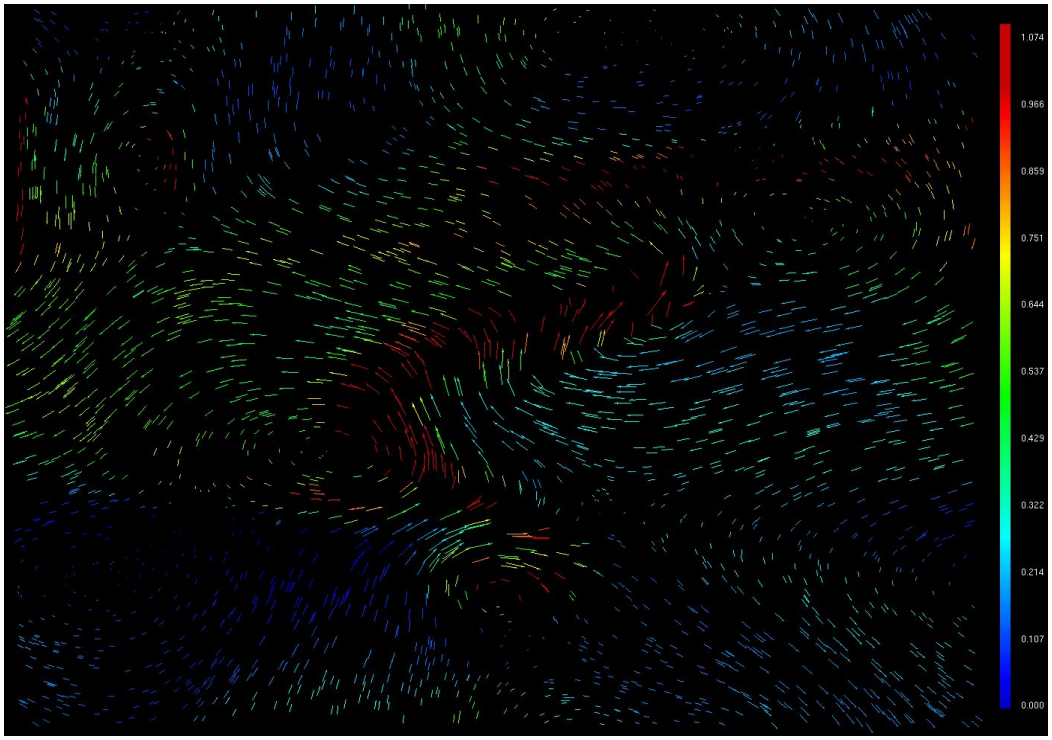

Image 3.3 - 50x50 jittered grid using arrow glyph and rainbow colormap based on the fluid density field.

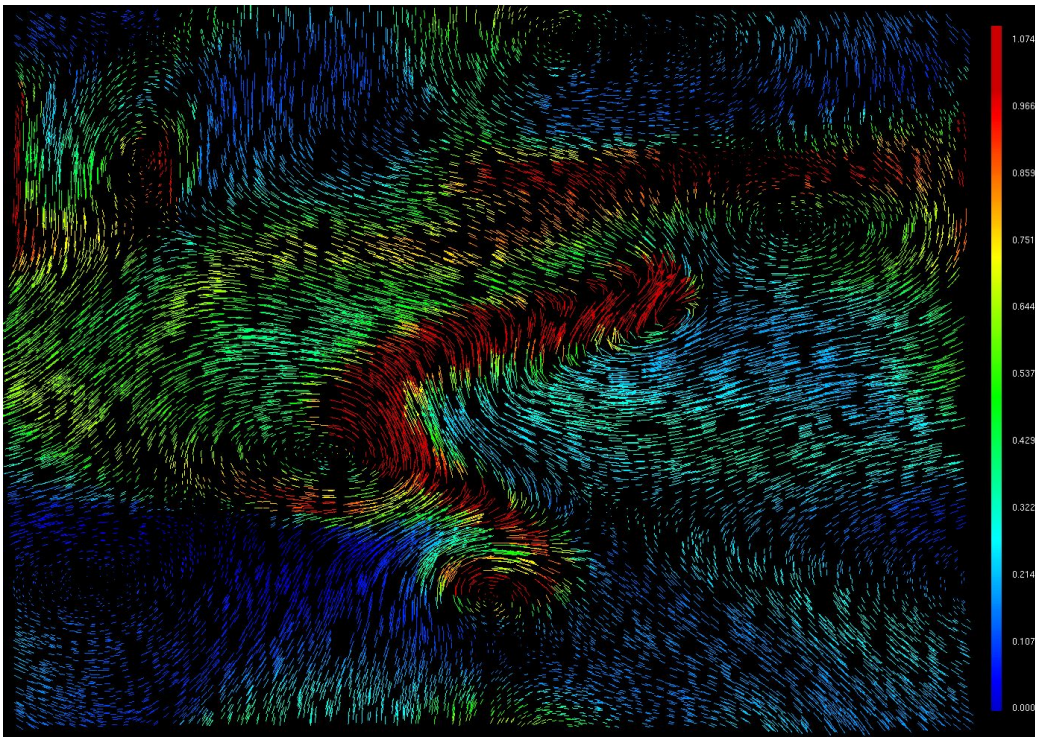

Image 3.3 - 150x100 jittered grid using arrow glyph and rainbow colormap based on the fluid density field.

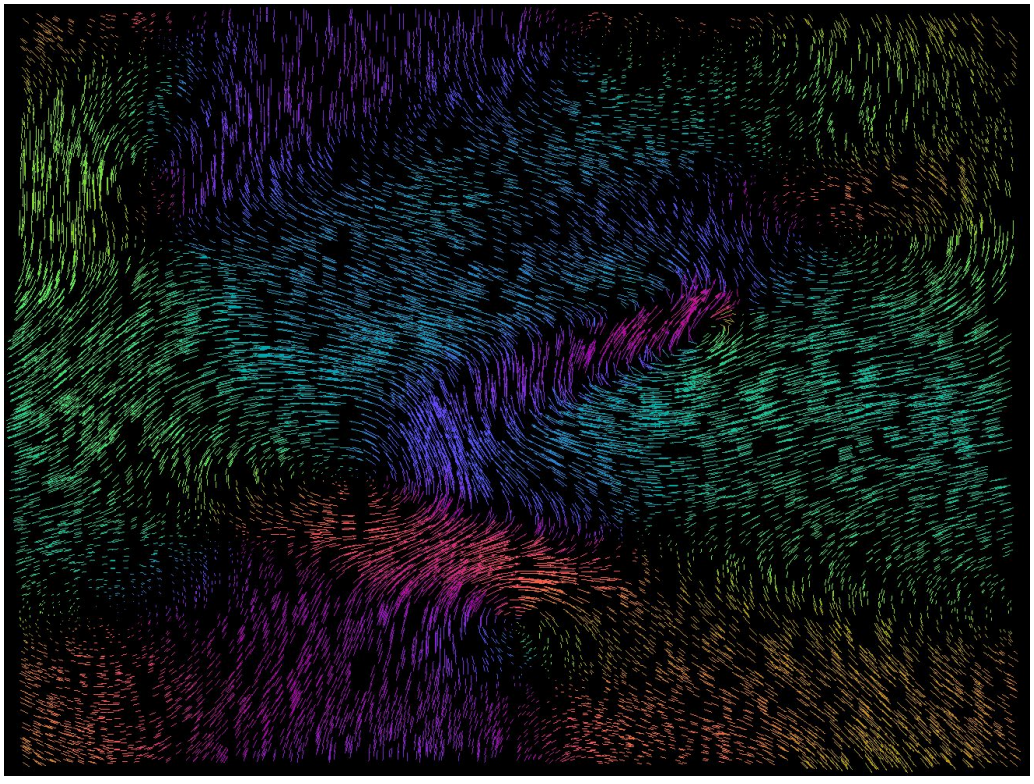

Image 3.4 - 150x100 jittered grid using arrow glyph and direction color mapping.

Vector glyphs can be a delicate technique in the sense that it offers trade-offs on the choice of parameters that will decide how much insight we can get on the studied field. The number of samples, their placement and their scaling factor (if any) will determine how clear and how useful the resulting image will be. If we use too many samples, we'll depict more data, but we are also bound to create more clutter if we are not careful about the scaling factor. If we use big glyphs, we can more easily represent the high magnitude areas and their direction, but we might occlude data, missing important small scale events, and create clutter.

Step 4: Divergence

Given a vector field, in our case the velocity and force fields, the divergence operator produces a scalar field that represents the amount of mass which is compressed or expanded whilst flowing the field.

For a vector field v: $R^3 \rightarrow R^3$, the divergence of the vector field $\mathbf{v} = (v_x, v_y, v_z)$ is given by

$$div\,\mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}.$$

To better illustrate the concept we will use the two images below. The first one displays the fluid density scalar field of a clock-wise vortex using the rainbow colormap. The second image shows the divergence of the velocity field colored with a diverging colormap.

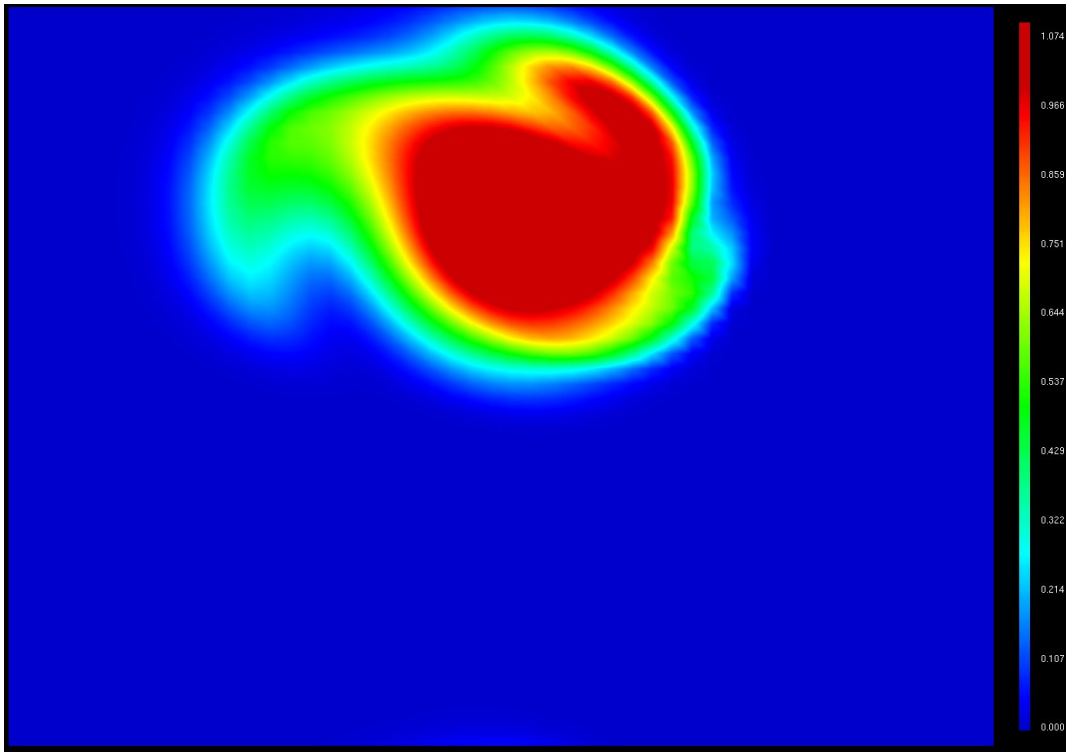

Image 4.1 - Clockwise vortex using rainbow colormap.

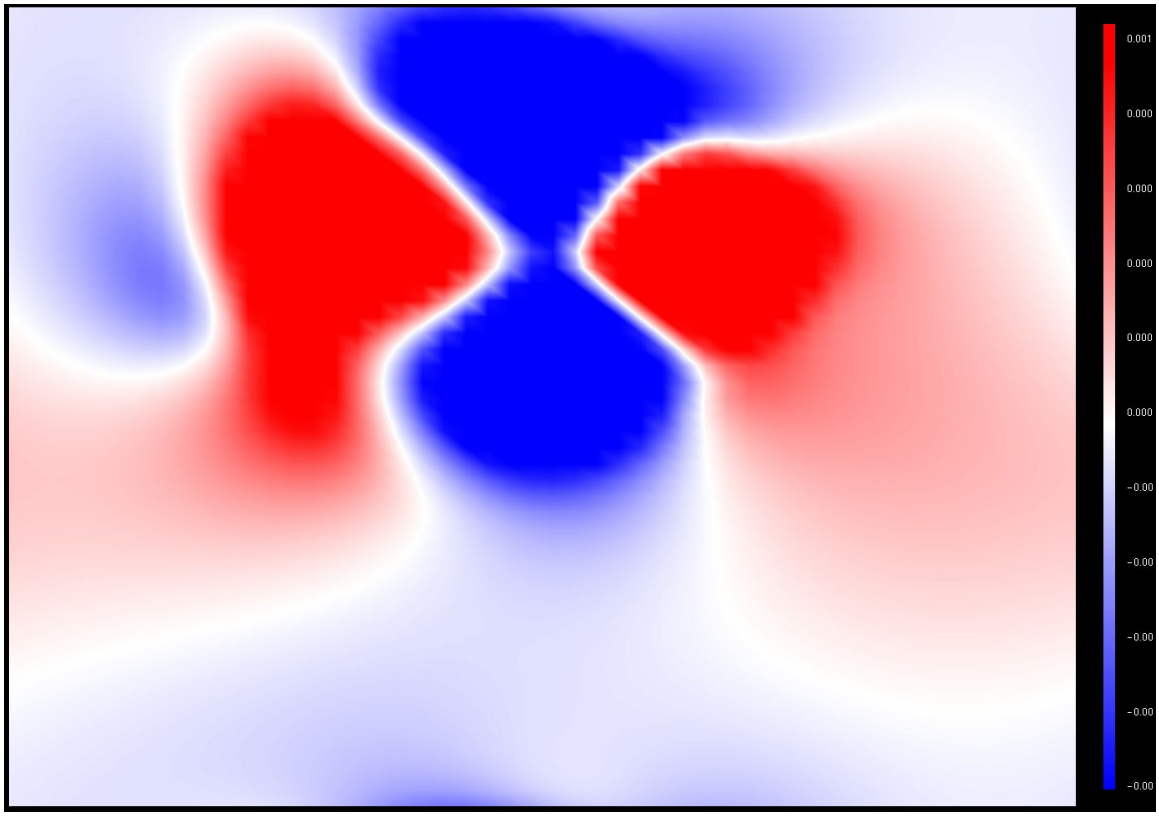

Image 4.2 - Divergence of the velocity field colored with a run-time defined diverging colormap clamped between the interval (-0.001, 0.001).

The red areas in the image have a positive divergence, intuitively, this means that mass spreads outwards from these points, characterizing them as sources. The blue areas represent negative divergence areas, meaning that these points behave as sinks, i.e. they suck the mass. White areas have a zero divergence, meaning that mass is being transported without expanding or compressing.

Image 4.2 gives us interesting insight on how the movement in the vortex is generated. Imagine the blue areas as being low pressure areas and the red as being high pressure areas. Fluid flows from red to blue, and the configuration seen above makes it move in a circular manner, creating a vortex. Because the data is continuous, between these different areas there is a region where the fluid is transported without being spread or sucked, these are the white areas. Thanks to texture based color mapping we don't get pink areas, which could cause confusion when interpreting the visualization.

The image below shows the behaviour of the force vector field. On the left the force magnitude scalar field is displayed with white glyphs indicating the direction of the force. On the right is the divergence field colored with the same colormap defined in Image 4.2.

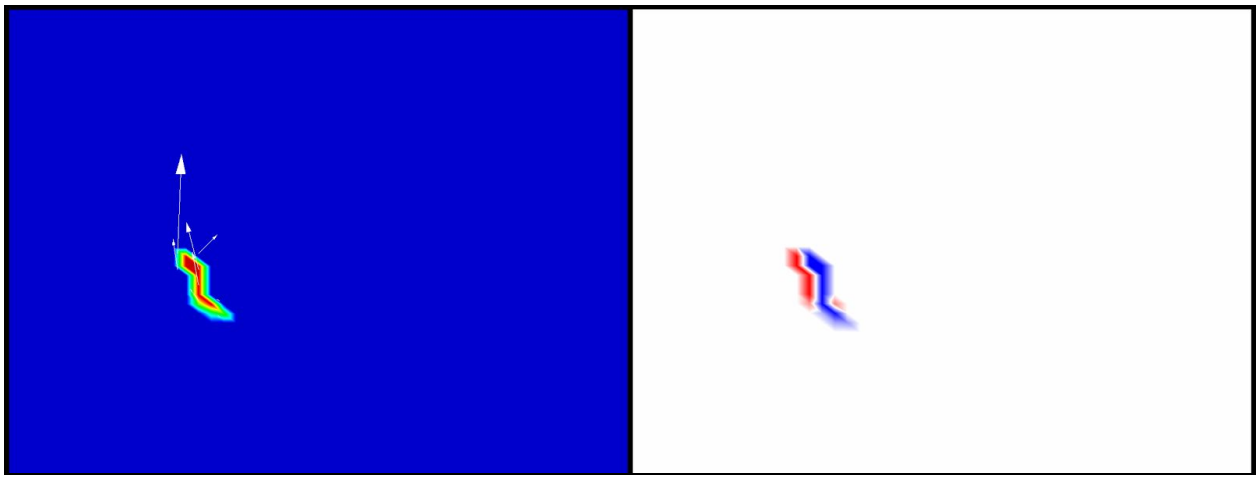

Image 4.3 - Magnitude and divergence of the force vector field that steers the simulation.

Step 5a: Isolines

Isolines or contour lines were used in cartography with the intention of indicating on land maps points that have the same altitude. This concept can be extended to any continuous scalar dataset, resulting in the highlight of points with the same scalar value.

We can implement isolines using a delta Dirac colormap, by having a small interval that “escapes” the current colormap and uses a distinct color, as illustrated in Section 2. In the example below, we have a conventional grayscale colormap ranging from 0 to 1, that in addition, in the interval 0.30 - 0.31 is colored in red. We can see the result of applying it to a scalar field in Image 5.2.

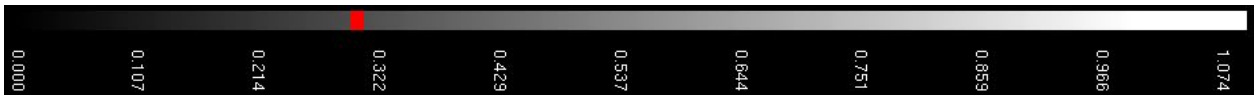

Image 5.1 - Resulting colormap configuration.

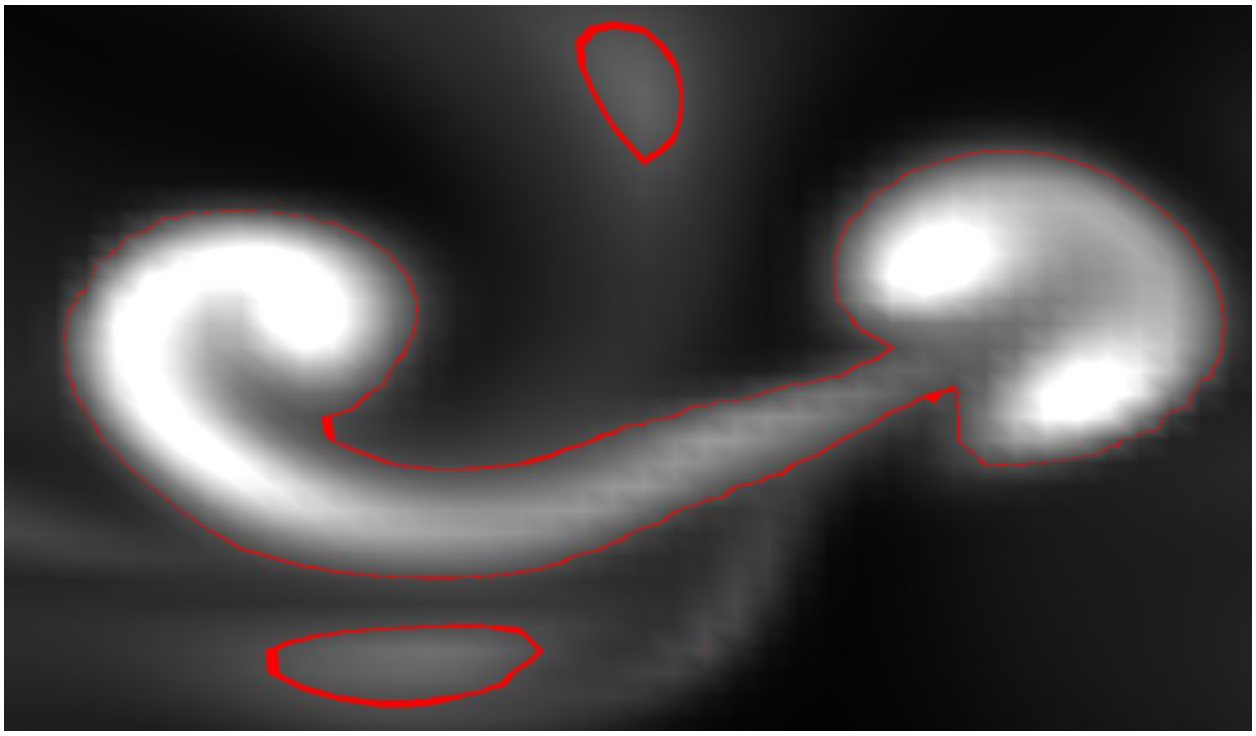

Image 5.2 - Result of the colormap applied to a fluid density field.

In Image 5.2, it is possible to see that the lines are not very well defined, in some places they are very thick bands and in others they can be barely visible. A superior result can be achieved via constructing isolines in a cell-by-cell fashion, by computing the points in the cell edges where the isovalue is reached (via interpolation of the cell vertices values) and tracing lines in these segments. The result of this technique can be seen on Images 5.3 and 5.4.

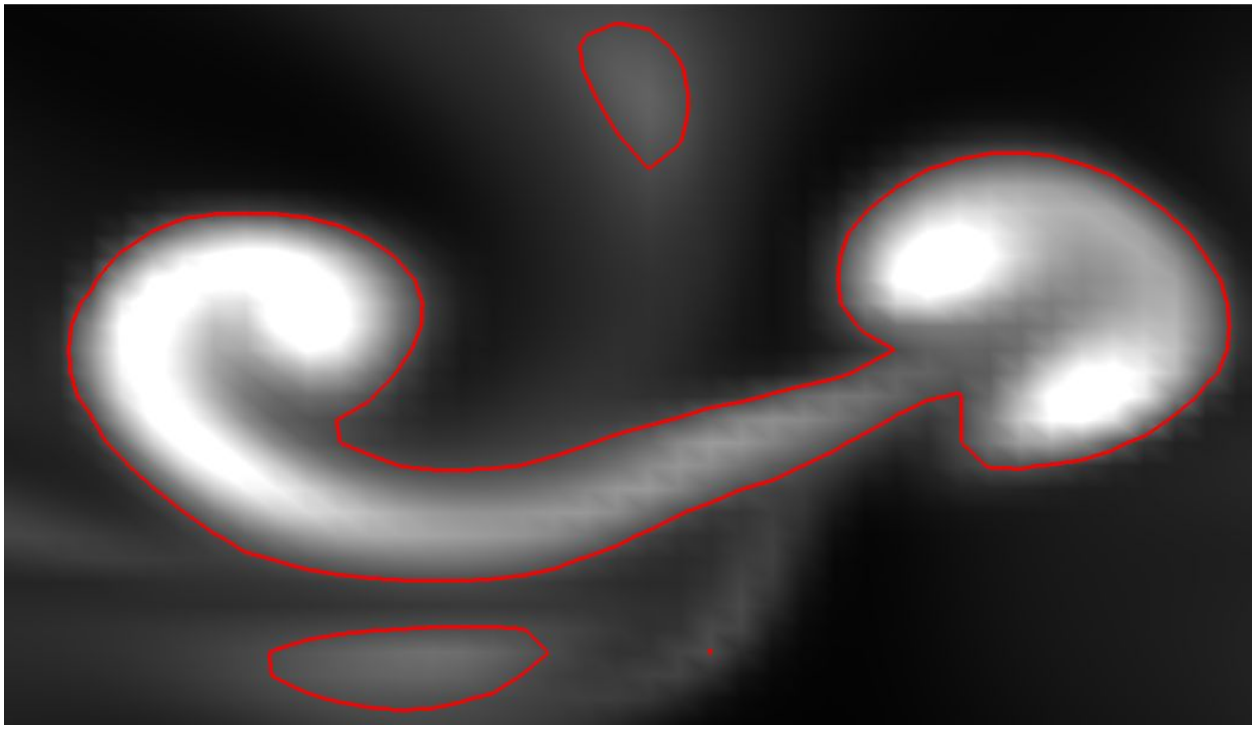

Image 5.3 - Result of isoline tracing for s = 0.3

The mechanism that manages the isolines has four parameters: a minimum scalar value, a maximum scalar value, the number *n* of isolines that are to be drawn between the min and max values, and the colormap which will be used.

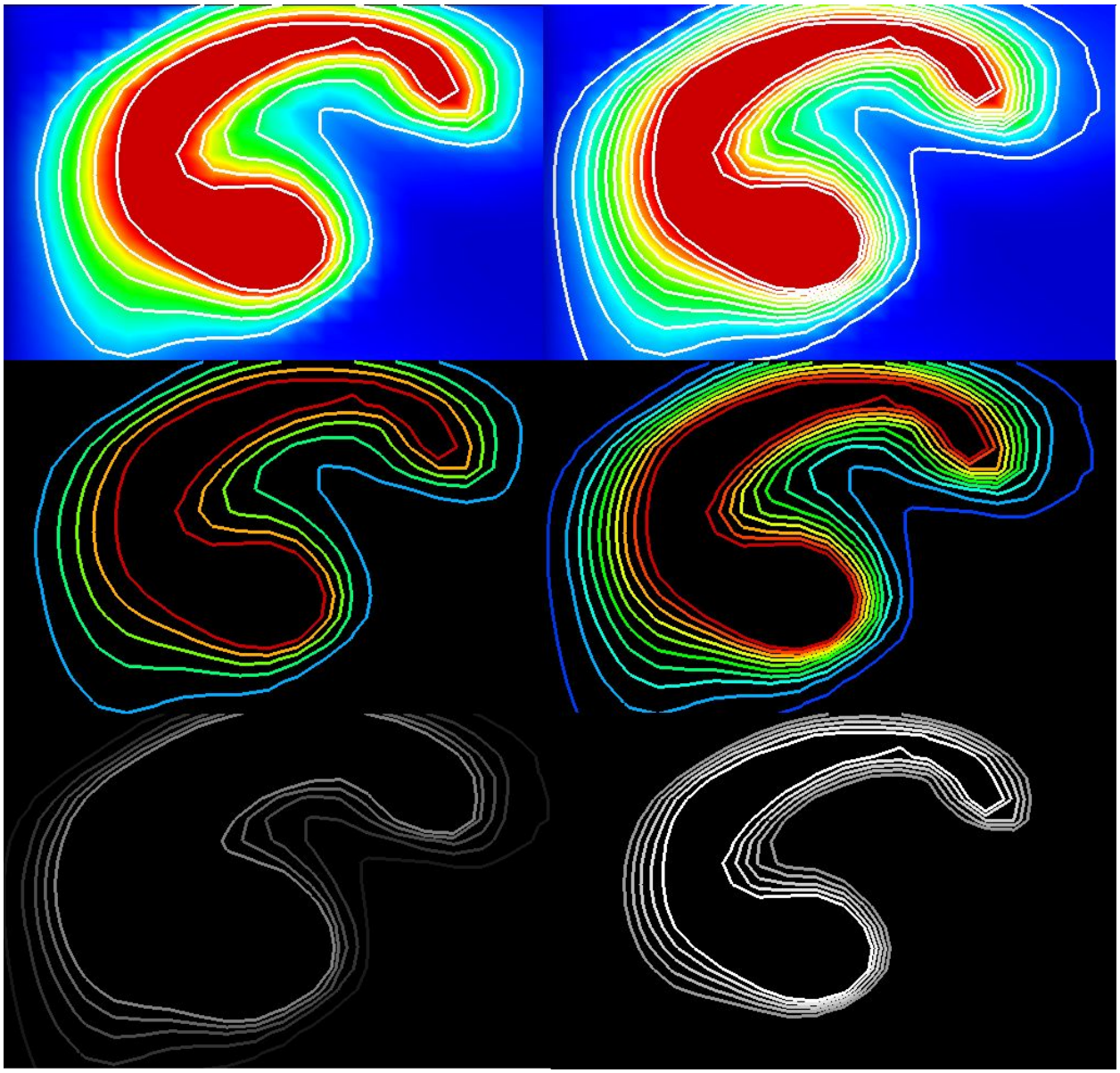

Image 5.4 - A: Min = 0, Max = 1, n = 5 with white colormap applied to isolines over rainbow colormap scalar field; B: Same as A, with n = 10; C: Min = 0, Max = 1, n = 5 using rainbow colormap; D: Same as C, with n = 10; E: Min = 0, Max = 0.5, n = 5 using grayscale colormap; F: Same as E, with Min = 0.5 and Max = 1.

Isolines are an interesting techniques because they give insight about the direction of the gradient of a scalar field - which is always perpendicular to the field's contour lines - and the slope of the field - by how close to each other the lines are.

Step 6a: Height Plot

Height plotting is an intuitive technique that uses the z axis to visualize 2D scalar fields. The height of the plot is defined in each point by the scalar value in that position. In the image below we can see the fluid density field displayed using this technique.

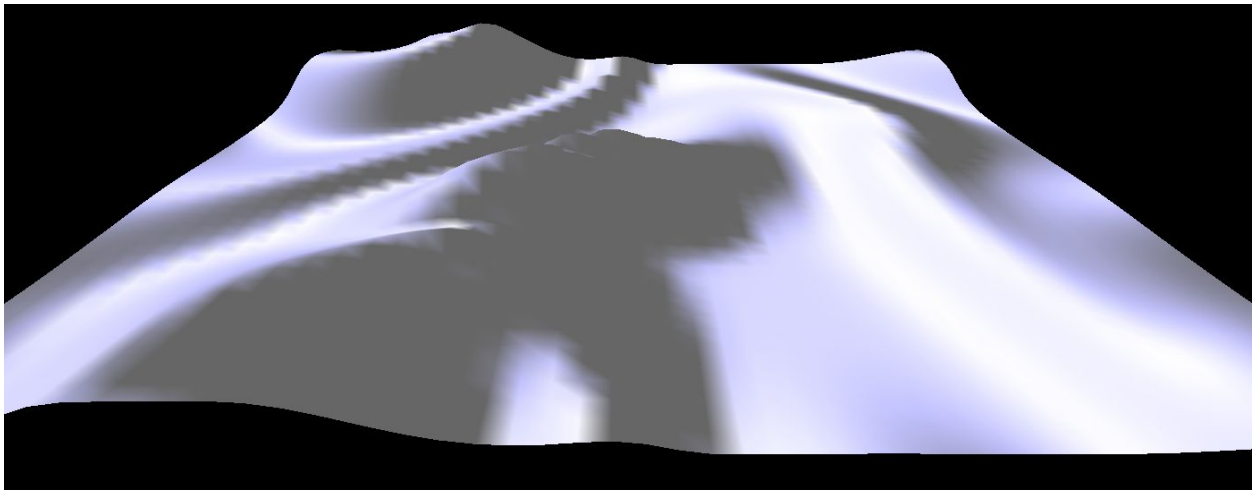

Image 6.1 - Height plot of the fluid density field.

It is possible to see that some lightning artifacts are present in the image. One of the reasons for this is the small 50x50 grid we're using. Increasing the size of the grid would alleviate these shadow artifacts.

Additionally, all color mapping techniques discussed in Section 2 can be applied to height plots.

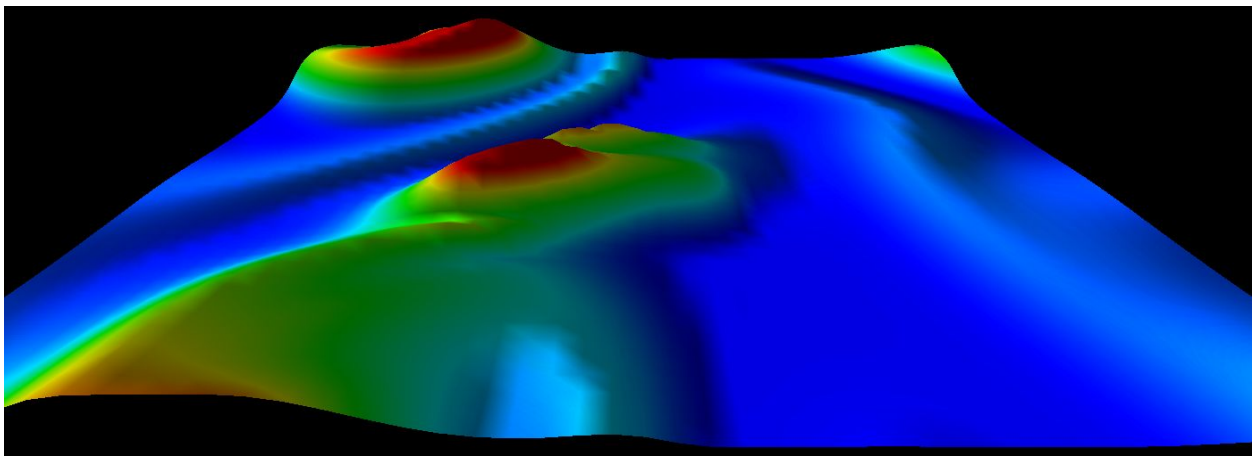

Image 6.2 - Height plot of the fluid density field colored with the rainbow colormap.

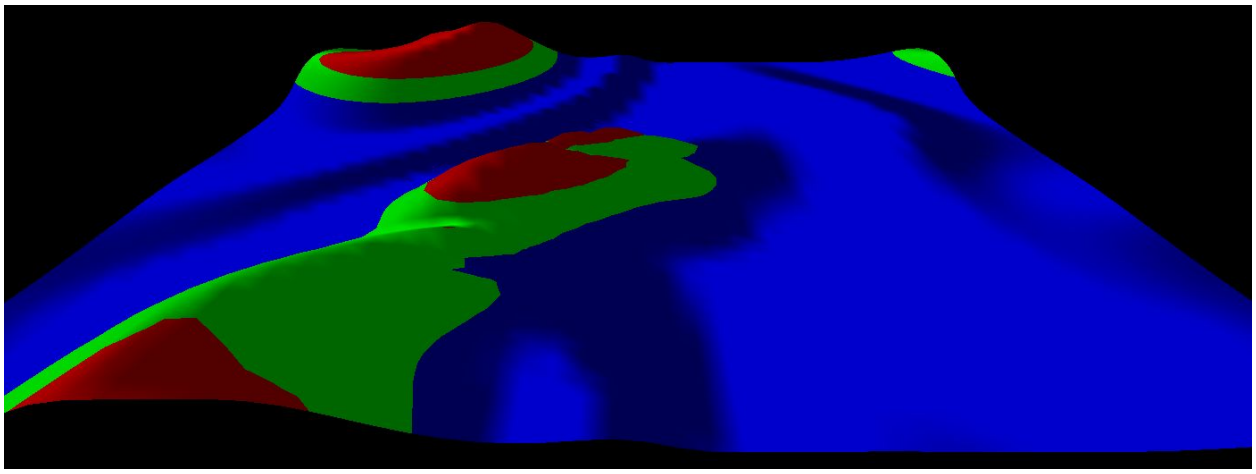

Image 6.3 - Height plot of the fluid density field colored with the rainbow colormap quantized to 3 colors.

The camera is free to move vai sliders on the UI.

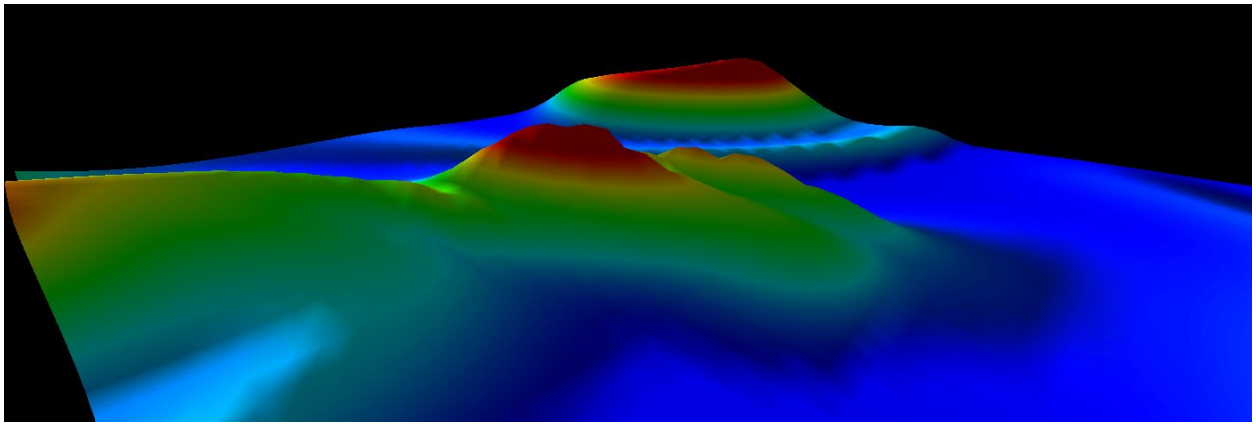

Image 6.4 - Height plot of the fluid density field with different camera parameters.

The technique can be also applied to the velocity magnitude and force magnitude scalar fields.

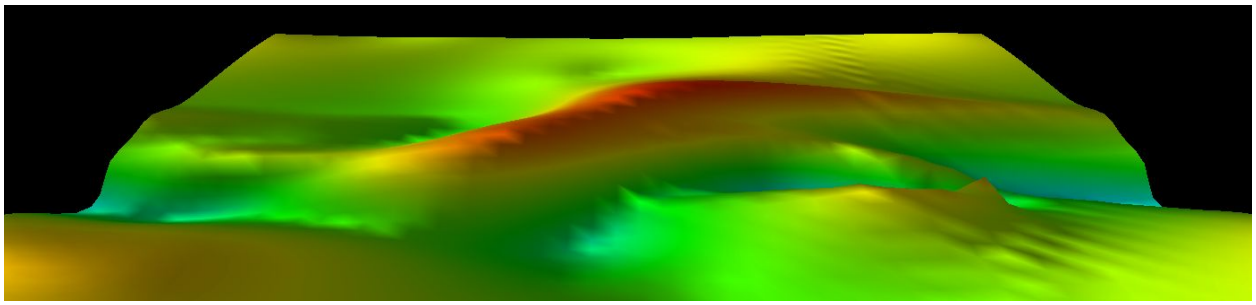

Image 6.5 - Height plot of the velocity magnitude field using the rainbow colormap.

Step 7a: Stream tubes

Stream tubes are used to visualize the trajectory of a particle in time, typically in a 3D vector field. In our case, as we are dealing with 2D vector fields, we will encode time in the z axis.

The way the algorithm works is: First, an arbitrary number *n* of vector field states that preceded the current state must be kept in a data structure - a queue is a very natural choice. Second, seeds must be placed in the oldest dataset. The way these seeds are placed is via mouse selection. Third, using Euler integration, we find out where the seed would be transported to if let loose on the vector field. This step is then repeated for all next 'slices' using the result of the last computation. Once we have all segments defined, we draw cylinders connecting them, incrementing the z value for each new point. The result of this technique can be seen on the next images.

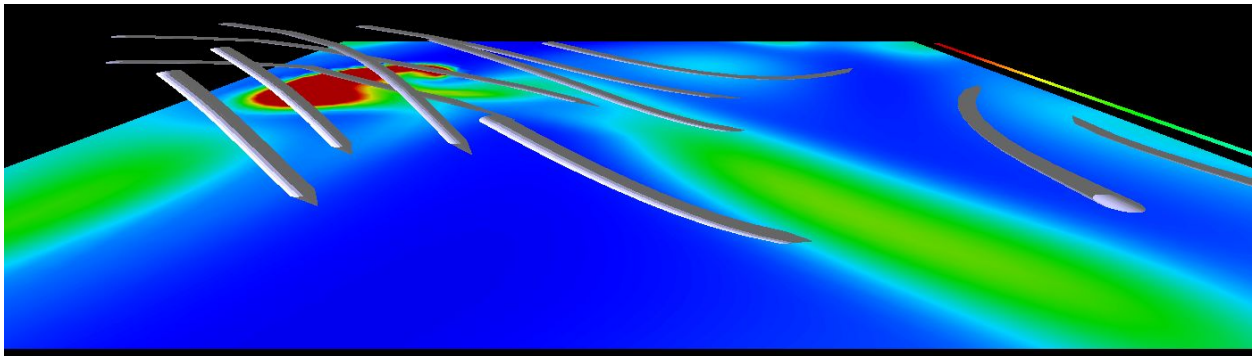

Image 7.1 - Stream tubes of fixed diameter based on the velocity vector field placed over the fluid density scalar field.

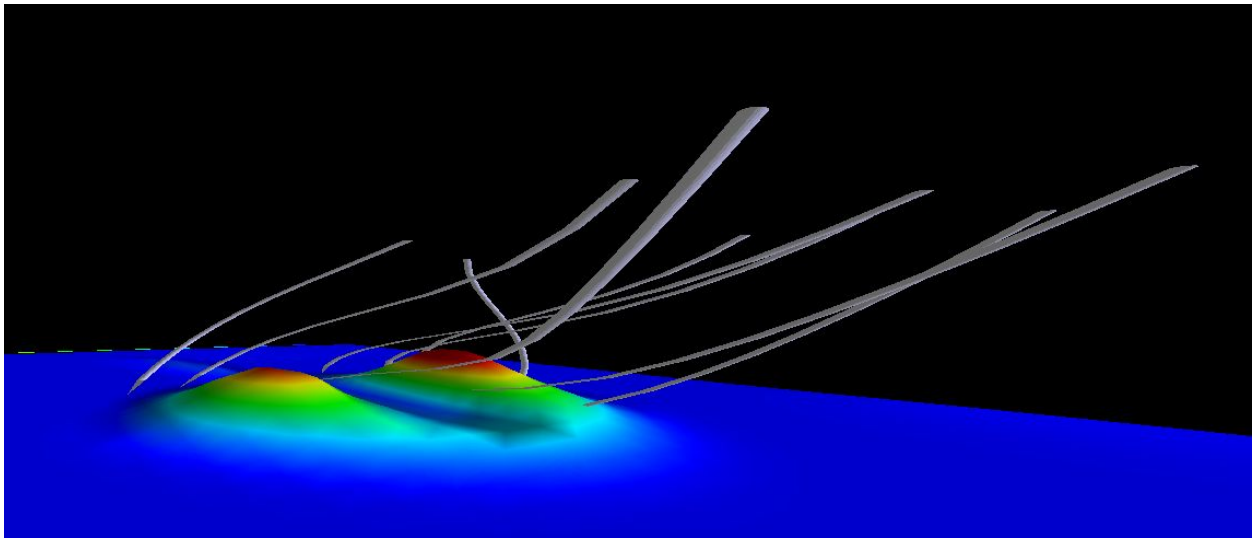

Image 7.2 - Stream tubes of fixed diameter over a height plot of the fluid density field.

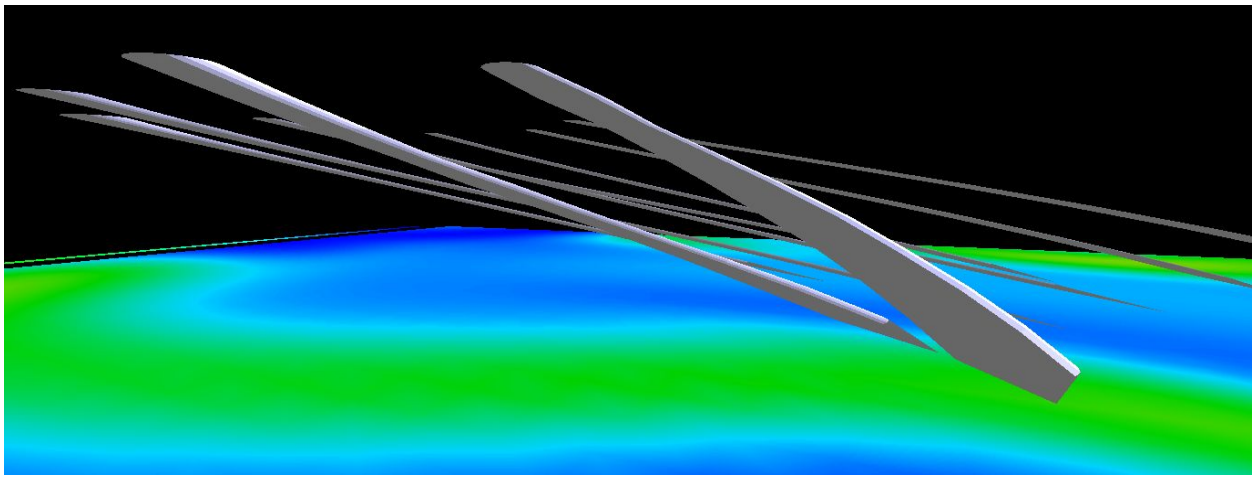

Figure 7.3 - Stream tubes with cross-section diameter matching the fluid density field. Notice how the closest tubes - that are in the green area - are thicker than the ones on the back - on blue area.

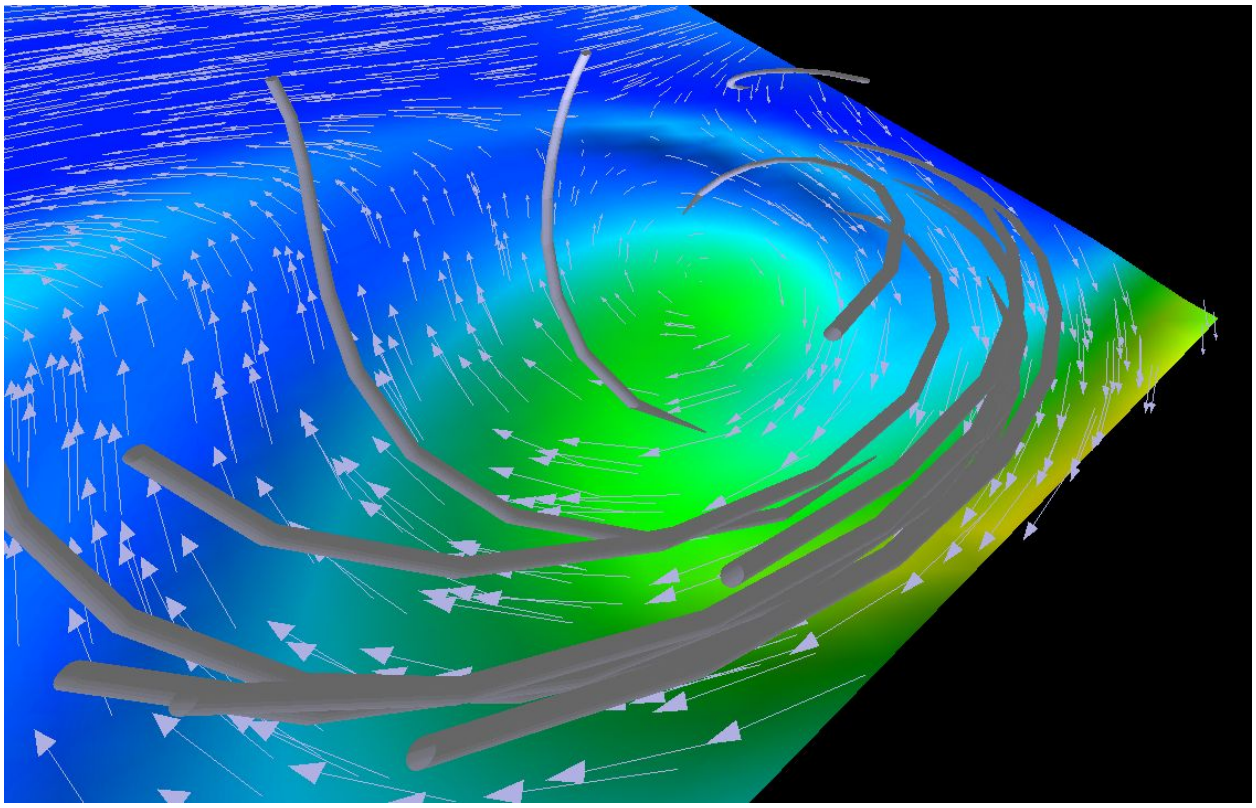

Figure 7.4 - Stream tubes on top of arrow glyphs - both representing the velocity vector field.

Conclusion

This was a very interesting project, it is quite amazing how from a 50x50 simulation grid, using mostly simple techniques, we can get insight in so many different aspects of fluid behaviour. It was very illustrative on the importance, difficulties and trade-offs present in each technique.

All the concepts implemented in this projected were well discussed in class, and the difficulties presented in the development of the project were not conceptual, most of them were related to openGL and programming issues.

The literature and extra material were very pertinent and the teacher assistants very helpful.

References

Telea, Alexandru C. Data Visualization. Hoboken: CRC Press, 2014. Print.

Shreiner, Dave. Opengl Programming Guide. Upper Saddle River, NJ: Addison-Wesley, 2008. Print.

Starn, Jos. "A Simple Fluid Solver Based On The FFT". Journal of Graphics Tools 6.2 (2001): 43-52. Web.