# Scientific Visualization

Amey Bhole (s3411427)
Steven Farrugia (s3362930)

Groningen, February 2019

## Contents

## 1 Introduction

The purpose of this project was for us to understand and implement a number of visualization algorithms to a fluid flow simulation - allowing the end user to interact and filter the visualization to their desired goal via a Graphical User Interface (GUI). This would inevitably help the end user gain further insight into the fluid simulation.

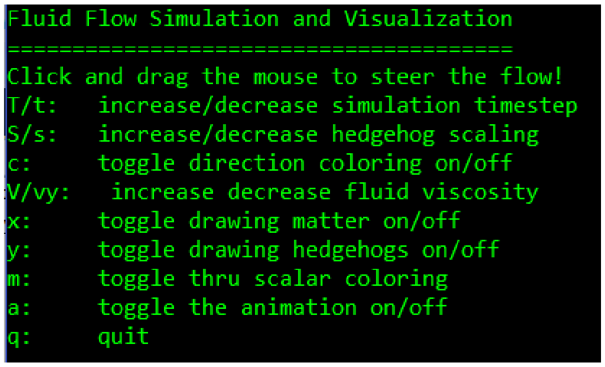
All development was carried out with C++, GLUI for GUI development and Linux OS. System specifications include a core i7 7700HQ with 16GB DDR4 ram.

## 2 Step 1

The purpose of step 1 was to compile the skeleton code provided. Having downloaded the linux project code from: http://www.cs.rug.nl/svcg/VIS/AssignmentCode, we compiled the project from terminal explicitly defining the file dependencies. This produced an executable file - 'Smoke.exe'.

### 2.1 Smoke.exe execution

Executing the Smoke.exe file produces two panels; Instructions and Visualization.



```
Fluid Flow Simulation and Visualization
=======================================
Click and drag the mouse to steer the flow!
T/t:   increase/decrease simulation timestep
S/s:   increase/decrease hedgehog scaling
c:     toggle direction coloring on/off
V/vy:   increase decrease fluid viscosity
x:     toggle drawing matter on/off
y:     toggle drawing hedgehogs on/off
m:     toggle thru scalar coloring
a:     toggle the animation on/off
q:     quit
```

Figure 1: Fluid Flow Simulation and Visualization instructions

Using mouse click and dragging along the visualization panel, we are able to produce smoke like visualizations as may be seen in figure 2. Scalar coloring and hedgehog was toggled on and hedgehog scaling was decreased prior to the mouse click and drag.

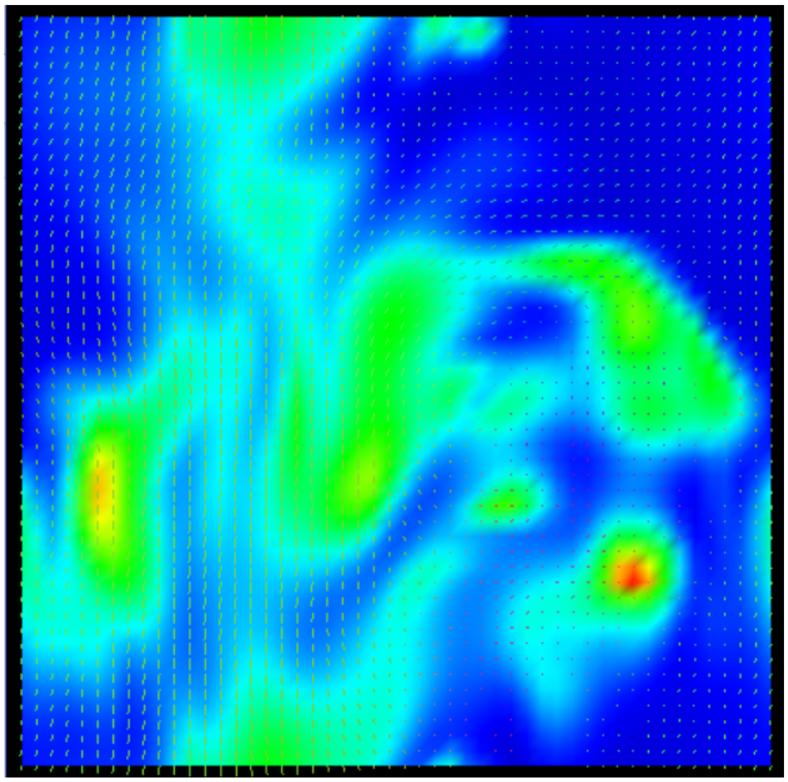


Figure 2: Visualization produced

To allow the user to interact with the application, we developed a GUI using the GLUI library. The library contained all the necessary functionality required for the purposes of the project in a simple to understand manner.

## 3 Step 2 - Colormaps

Part 2 of the assignment entailed developing a set of color mapping techniques to be applied to the fluid simulation along which would also be applied to the three datasets; fluid density (rho), fluid velocity magnitude $||v||$ and the force field magnitude $||f||$.

The velocity and force fields were computed on the fly - taking the euclidean distance of the selected velocity or force datasets in the x and y direction for each data point.

### 3.1 Colormaps

Four color mapping techniques were implemented for the fluid flow simulation; Gray-scale, 1) Grayscale, 2) Rainbow, 3) red-to-yellow - 2 Hue and 4) the Zebra colormap. Each colormap has it's benefits and drawbacks which we will now define in the respective subsections below. The implementation has been made available to the user through the use of a drop down menu.

#### 3.1.1 Grayscale Colormap

The grayscale colormap is the simplest and easiest to follow - whereby scalar values $f$ are mapped linearly to a gray value. Mapping the smallest and largest $f$ value to black and white respectively. The relative simplicity of understanding such a colormap is both due to it's natural ordering factor and linear mapping to luminance. Although, differentiating between different gray values is harder when compared to other colormaps - such as those using hue.

#### 3.1.2 Rainbow Colormap

One of the most common and widely available is the rainbow colormap. It heightens our focus towards higher end values; warm (red) color for the largest scalar values in the dataset. Having an order of hues from blue to green to yellow to red, users may not be experienced and therefore need to learn such ordering. Another issue with the rainbow colormap is that some of the colors may appear to map non-linearly.

#### 3.1.3 Red-to-yellow - 2 Hue Colormap

Two-hue colormaps solve some of the issues present in the rainbow colormap. Using a black-to-red colormap, the low and high scalar values are mapped to red and yellow, respectively. The resultant map allows for an easy color ordering as well as a perceptually more linear result to that produced by the rainbow colormap. Similar to the gray-scale colormap, we also linearly interpolate between the two defined colors. The obvious limitation is the lack of dynamic range apparent in the rainbow colormap.

#### 3.1.4 Zebra Colormap

The Zebra map was chosen to highlight variations present within the data available; displaying regions with the highest or limited/no variability using thin or thick bands, respectively. Use of this colormap restricts the end user to variability in data and not in the direction with which the variability is maximal.

The aforementioned colormaps may be seen in the figure below.

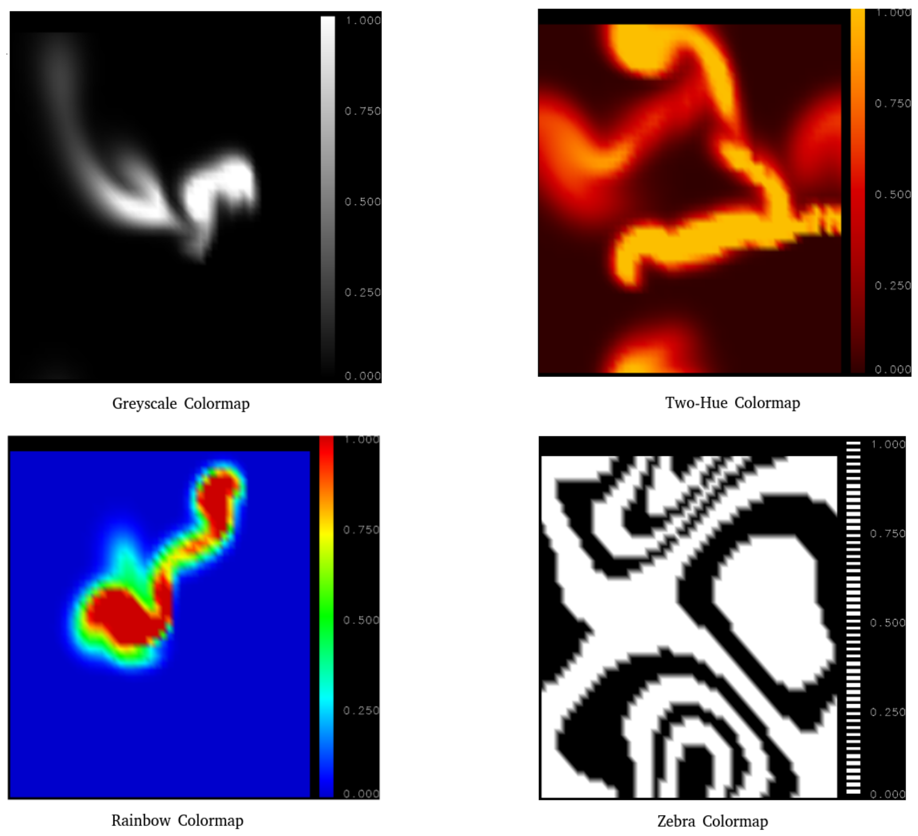


Figure 3: Colormaps used; Starting first row from left; Greyscale, Two-hue, Rainbow and Zebra colormap

## 3.2 Parameterizing the colormap itself

### 3.2.1 Color banding

Color banding allows us to limit the number of visible colors by quantizing, also known as binning, the scalar dataset available into the desired number of colors. This method is desirable when visualizing categorical datasets for which we would want to produce a color for each category. Using a low number of colors we effectively create sharp transitions in the colormap. This effect may be reverted if we increase the number of colors, creating smoother transitions in the colormap.

As we may see from figure 4, the scalar value has been binned into 5 colors.

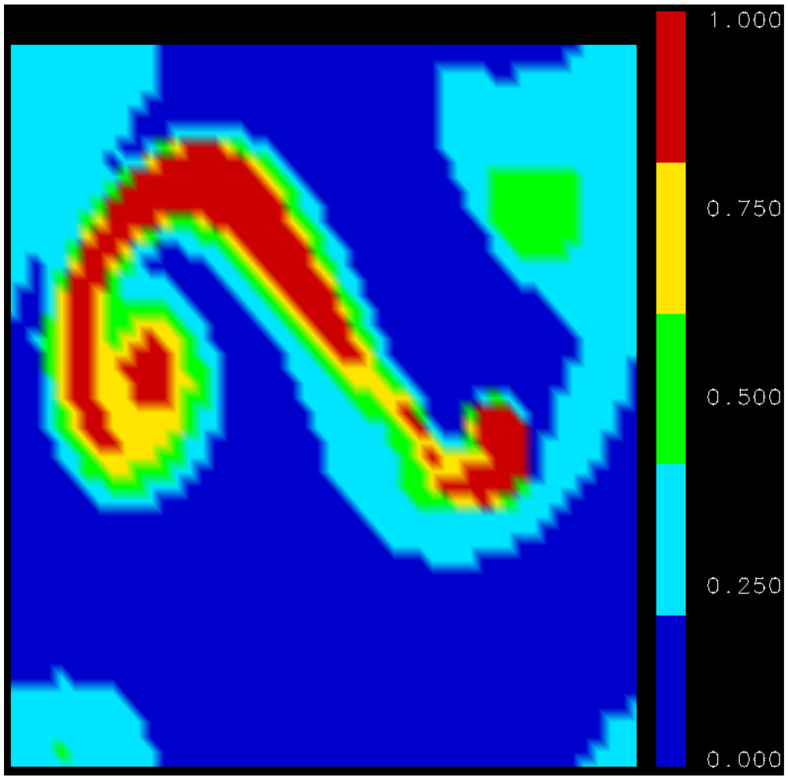


Figure 4: Defining number of colors to 5 using the rainbow colormap and rho dataset

### 3.2.2 RGB-HSV - Saturation and Hue

The HSV color standard is more convenient for the end user - giving the end user the option to alter the hue and saturation of the colors. We therefore need to convert the current colormap to HSV in order to allow the user to access the parameters. Both hue and saturation range between 0 and 1 with hue being set to 0 by default - representing the different colors on the different wavelength and the saturation value to 1 specifying pure colors. For Value parameter (also known as alpha) has been preset to 1 - defining the opacity. The colors are then converted back to RGB since some colors are misrepresented in HSV.

As an example, changing the hue and saturation parameters to 0.3 and 0.7 converts the rainbow colormap to the following;

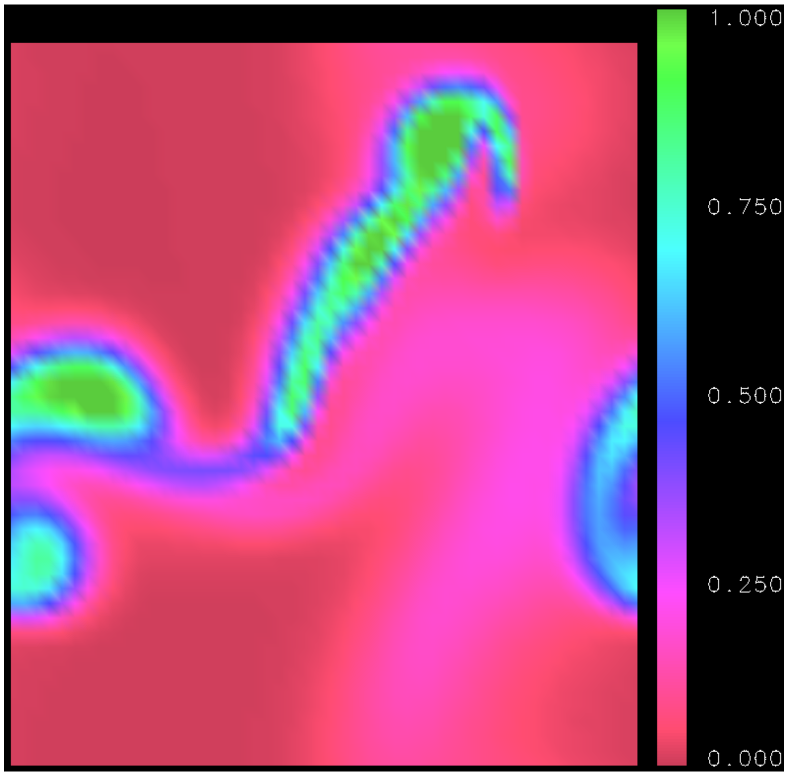
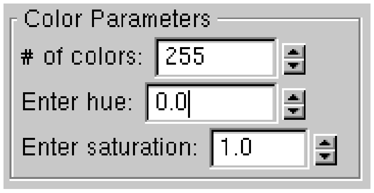


Figure 5: Hue set to 0.3 and Saturation 0.7 using rainbow colormap on rho dataset

The GUI encapsulating the above color banding and Saturation/Hue parameters;



Color Parameters

| | |
|---|---|
| # of colors: | 255 |
| Enter hue: | 0.0 |
| Enter saturation: | 1.0 |

Figure 6: Color Parameter panel

## 3.3 Scaling and Clamping

The user is provided with two options; scale and clamp. By default all scalar values are scaled between 0.0 and 1.0. In order to scale the value to the current dataset scalar value, we must calculate the current min and max values at each current time. The resultant computed min and max values at each time step, along with the mid value and mid-to-max and mid-to-min values are displayed alongside the color legend - placed relative to the corresponding color. Scaling may be toggled on and off from the GUI.

The user may also define the respective value(s)for which he/she may wish to clamp the min and/or max values. Any values which fall outside the scalar range $[f_{min}, f_{max}]$ are assigned to the min or max values, respectively. These are directly mapped to the entire range of the color legend. Two float fields, corresponding to the min and max clamp values have been provided as parameters in the GUI.

## 4 Step 3 - Glyphs

Using the techniques defined and implemented in Step 2 to represent the scalar values, we are unable to convey the direction, orientation and magnitude of a vector field using glyphs. A hedgehog implementation is provided by default within the skeleton code - but lacks the ability to convey the orientation or magnitude of the vector field being depicted. We therefore explain our implementation of 3D-shaded arrows and cones which are able to convey both orientation and angle.

### 4.1 Scalar and Vector datasets

One may represent any scalar datasets (fluid density, $||v||$ and $||f||$) using the color map techniques defined in step 2 with a selection of the vector fields (fluid velocity $\mathbf{v}$ and $\mathbf{f}$) to represent the represent the orientation and magnitude. This allows for various combinations, which may be desirable by the end user, to be visualized.

### 4.2 Interpolation

Three implementations of interpolation were defined; the default, Nearest Neighbour and Bi-Linear Interpolation. These allow us to compute the respective scalar value out of the neighbouring point(s) should the samples points fall outside of the grid points.

### 4.3 Glyph types

A total of 3 glyphs are available; the default oriented line glyphs (hedgehog - provided in the skeleton code), cones and arrows. Both the cones and arrows have been shaded, giving a 3D look.

The resultant implementation of the arrows and hedgehogs may be seen in the figure below while the cones may be seen in 9.

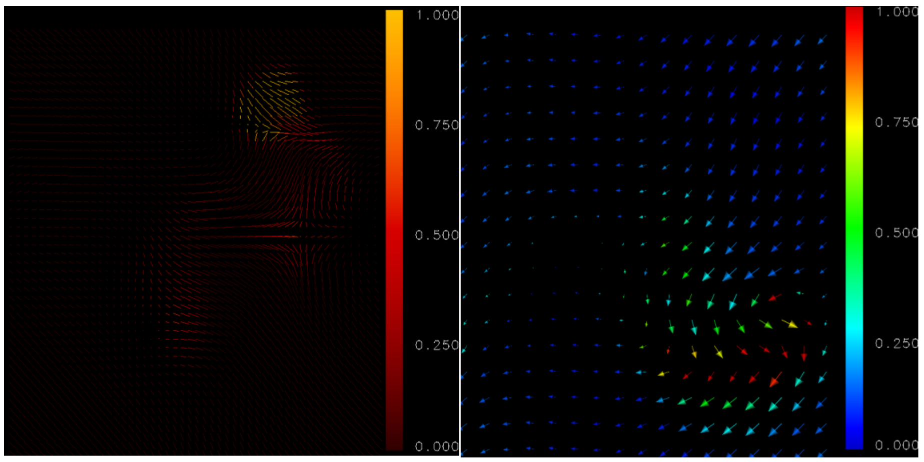


Figure 7: Arrows and hedgehog glyphs

## 4.4 Reducing clutter

Since a vector icon is drawn at every sample point of the dataset, the result is potentially a cluttered visualization which in effect makes it hard to understand or gain any insight as may be seen from figure 8. Therefore two options are made available; uniform sub-sampling and scaling the vector magnitude.

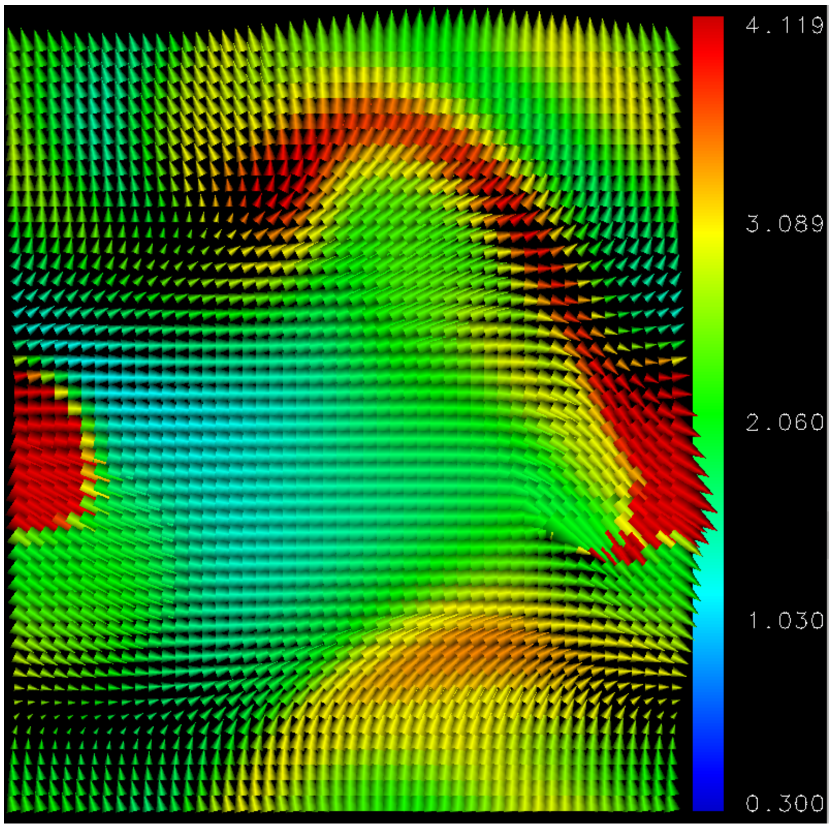


Figure 8: Glyph Clutter with no sub-sampling or scaling

The scaling functionality is made available by pressing the keyboard letter 's' or 'S' to scale down or up, respectively. This allows us to reduce the size of the glyph determined by the magnitude of the vector field being calculated. Even though this may reduce the visible clutter by not making the glyph icons overlap, this potentially makes the glyphs too small. Therefore, the optimal solution is a combination of scaling and sub-sampling the grid. Sub-sampling is provided as a GUI parameter - set to 50 by default. The resultant combination of the two functionalities may be seen in the figure below.

An alternative solution for scaling would be to use a nonlinear scaling term which varies non-linearly with respect the vector field to drop the direct 1-to-1 relationship present between the vector magnitude and glyph length.

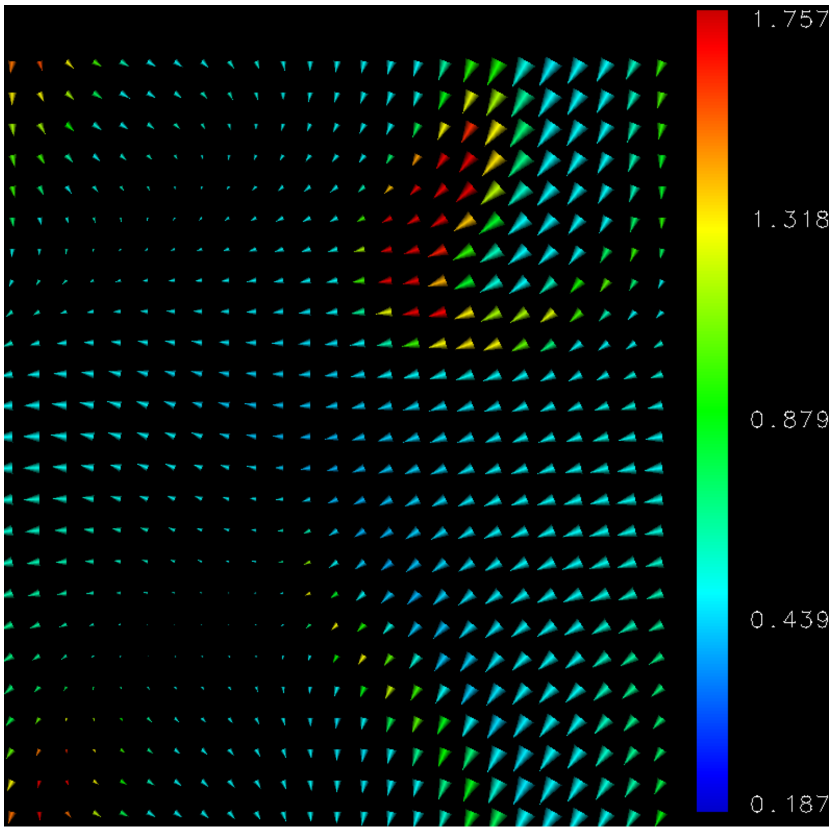


Figure 9: Glyph Clutter with sub-sampling or scaling

## 5 Step 4: Divergence

In this step, we use the the vector field operator to compute various quantities on vector fields for analysis on vector dataset. Divergence is a quantity used for vector field visualization. In particular, we implement divergence using the following formula:

$$div\ \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \tag{1}$$

The source and sinks occur when the divergence is positive and negative respectively. We implemented the divergence for fluid velocity v and force field f. Figure 10 shows the source and sinks represented by red and blue for rainbow color map. The application provides an option to visualize divergence for different scalar and vector fields shown in Figure 10 and 13 and can be used for different color maps. Figure 11 shows the divergence where the scalar field is force and vector field is velocity.

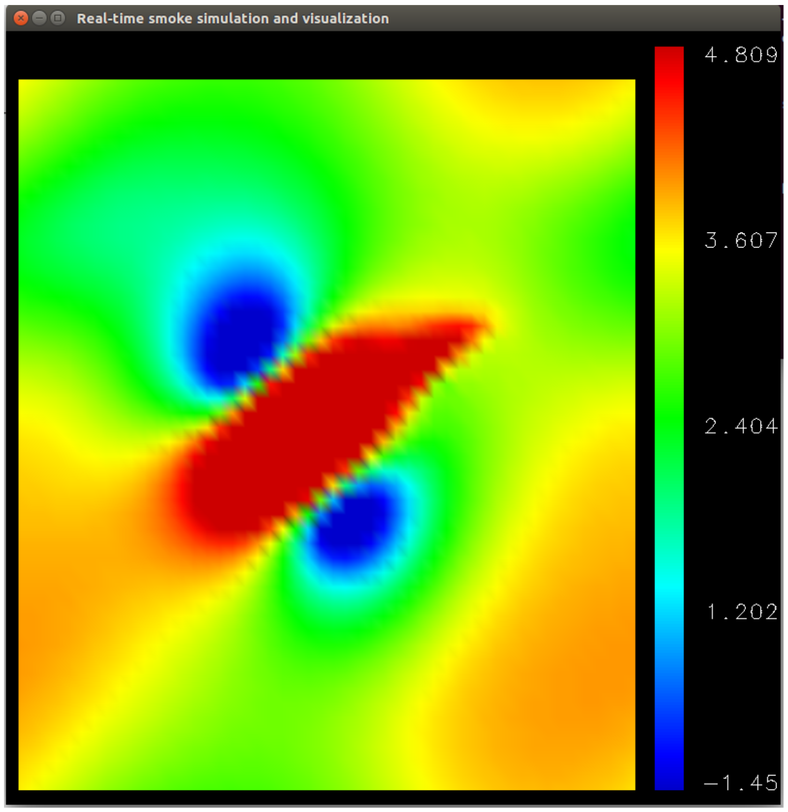


Figure 10: Divergence for fluid velocity v

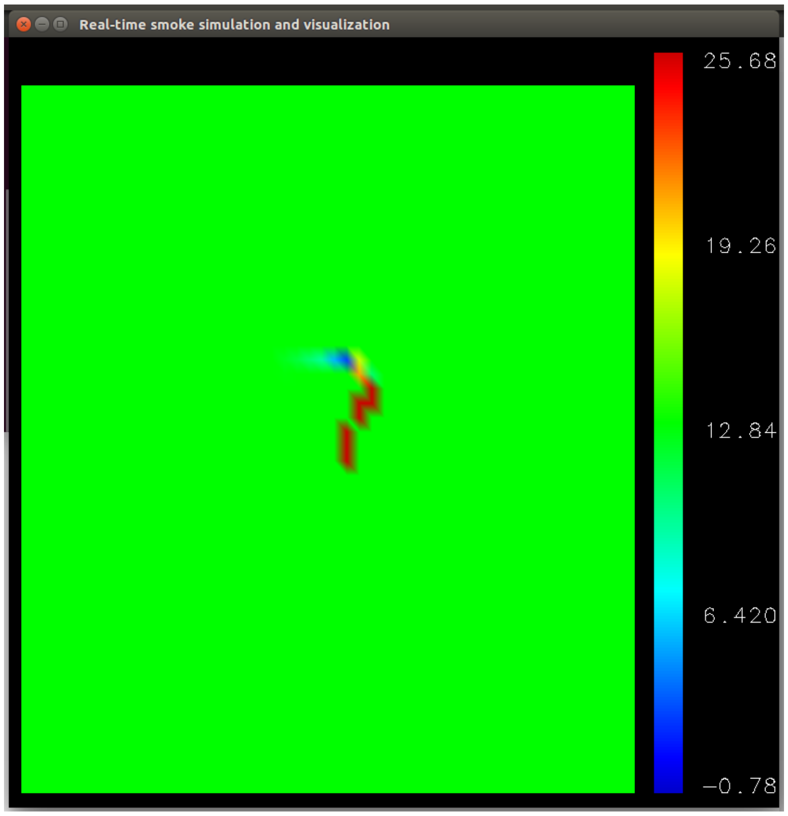
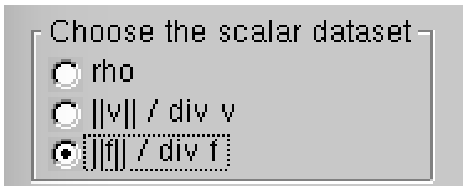


Figure 11: Divergence for force field f



rho
||v|| / div v
[✓ ||f|| / div f]

Figure 12: Choose scalar dataset

[✓ v]
f

Figure 13: Choose vector field

## 6 Step 5: Isolines

Isolines represents points of a scalar dataset's domain where the value equals to a threshold value. The isolines are visualized using the marching squares technique for the fluid density rho. The marching squares algorithm determines the topological state of each cell based on the isovalue provided as input. The cell's topological state shows which lines are intersected by the contour and the connections of the intersection with line to produce isolines.

A quad cell has $2^4 = 16$ different topological states. The state of the quad cell is given by a 4-bit integer index for storing inside/outside state of vertex. The integer index is used as a case table for holding optimized code for all 16 topological state. The application provides an option to draw the isolines and provide a input for the threshold. Figure 14 shows the isolines for threshold of 0.5.

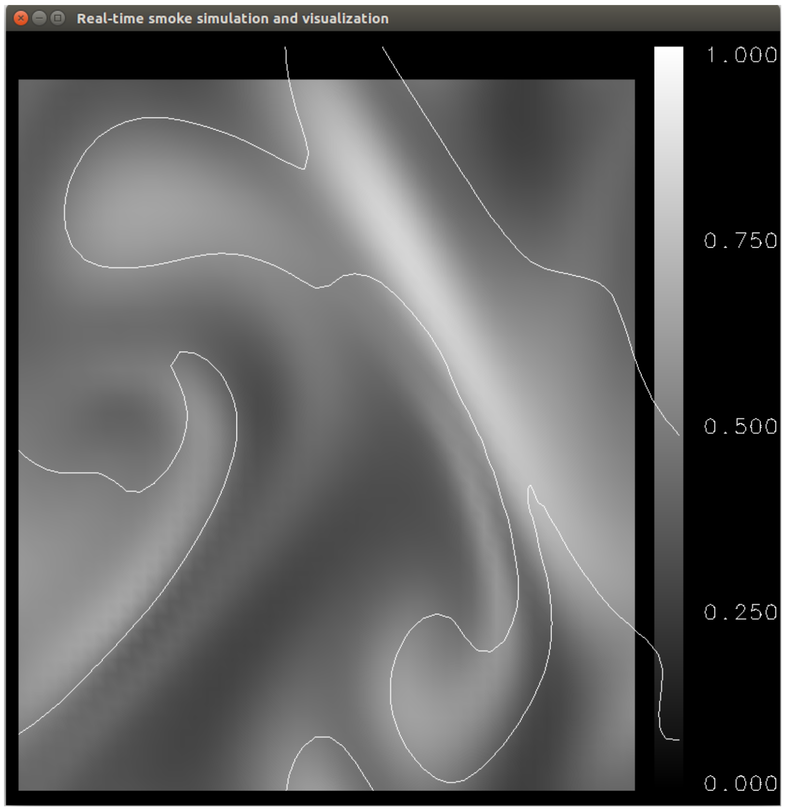


Figure 14: Isolines with threshold 0.5