University of Groningen
Faculty of Mathematics and Natural Sciences
Department of Mathematics and Computing Science
Scientific Visualization

Scientific Visualization

Real-time simulation of fluid flows

Christian Manteuffel
Spyros Ioakeimidis

1.0
Groningen, January 28, 2013

Contents

List of Figures

1 Introduction

2 Implementation

2.1 Skeleton compilation

use the standard version recompiled the lib for x64 os x changing to framework different headerfiles

2.2 Color mapping

Colormapping is one of the most straight-forwards ways to visualize scalar values. This visualization technique maps each value to a specific color by using a scalar-to-color function (cf. Figure 3).

We implemented this function as a lookup-table because we think that is is easier to define custom colormaps than compared to a transfer function. A disadvantage is that the lookup-table probably is more complex because it requires additional methods to manage the table.

We implemented the lookup-table as an array of 256 RGB color values. RGB has the advantage over HSV that it is additive, which makes linear interpolation easier. A colormap is defined by specifying a color at various points in the array. All other colors will be calculated by linearly interpolating between the specified color-points. For instance blue at index 0, yellow at index 127 and green at index 255. This will produce a colormap similar to the one shown in Figure 2(d).

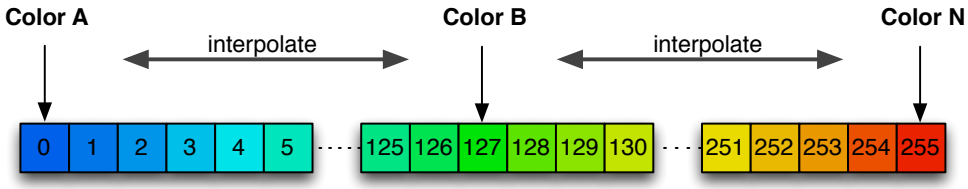

Figure 1: Conceptual model of the lookup-table. Missing colors are calculated by interpolating defined color-points.

In the current implementation the colormaps are defined programmatically but it would be possible to easily create a user interface that lets the user define his own colormaps during runtime via dran'n'drop. The following colormaps have been predefined as illustrated in Figure 2:

(a) **Luminance (grayscale)** A simple universal colormap that adheres the linearity constraint.

(b) **Rainbow** A colorful colormap that draws attention to high values as they are displayed in red. However, this colormap is often considered confusing due to unclear perceptual ordering of values and obscurity of data [1]

(c) **Heat** This type of colormap has a natural association with flames. Very high "hot" values are bright while lower values range from black to dark red.

(d) **Blue-Yellow-Green** A colormap that separates the range of values into low, middle and high values.

(e) **Black Gradient** A colormap that produces a gradient from any color to black.

(f) **White Gradient** A colormap that produces a gradient from any color to white.

[1] Rainbow Color Map (Still) Considered Harmful, David Borland and Russell M., IEEE Computer Graphics and Applications

(g) **Zebra** A colormap that shows rapid value variations. This colormap makes it difficult to map certain regions to concrete values as low values cannot be distinguished from high values.

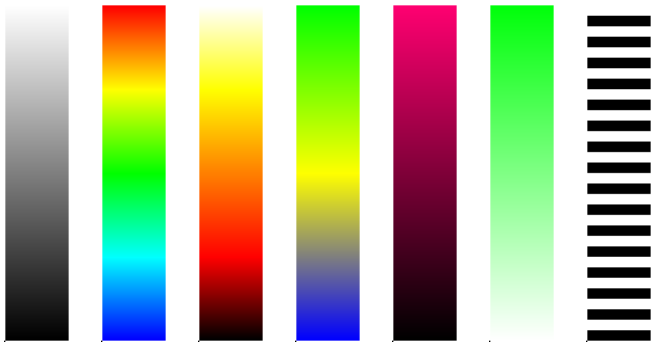

Figure 2: Predefined colormaps: (a) Luminance, (b) Rainbow, (c) Heatmap, (d) Blue-Green-Yellow, (e) Black Gradient (f) White Gradient (g) Zebra

Furthermore the user has the possibility to change hue and saturation of the colormap during runtime. In order to apply hue (h) and saturation (s), the colors are translated into the HSV color system. Due to the circular nature of the hue, h shifts the color(c) along the hue color wheel. Instead saturation is applied by multiplication, which implies that the saturation can only be decreased for any color. A less saturated and hue shifted version of the rainbow colormap is illustrated in Figure 4.

$$\begin{aligned} c_{hue} &= (c_{hue} + h) \ \% \ 1 \\ c_{sat} &= c_{sat} * s \end{aligned}$$

three datasets: The fluid density rho, the fluid velocity magnitude | v | fig:velocityZebra, and the force field magnitude | f | 9

how to specify a colormap parameterization of colormap (hue and saturation) 4

defined colormaps 2

vertex vs texture

transform (linear or logarithmic)

banding

legend The legend should show both the colormap and the corresponding numerical values associated to various colors.

scaling: the entire actual range (min..max) of the dataset at the current time moment is mapped to the visible colormap. This means, of course, that you have to update the numerical values displayed along with the color legend. 9

clamping: the data values are clamped between two user-prescribed min and max values. These values correspond to the entire range of your color legend. Actual data values higher than min, or lower than max, are actually clamped to min and max respectively. The user should be allowed to change min and max interactively, e.g. by means of some sliders or similar widgets.

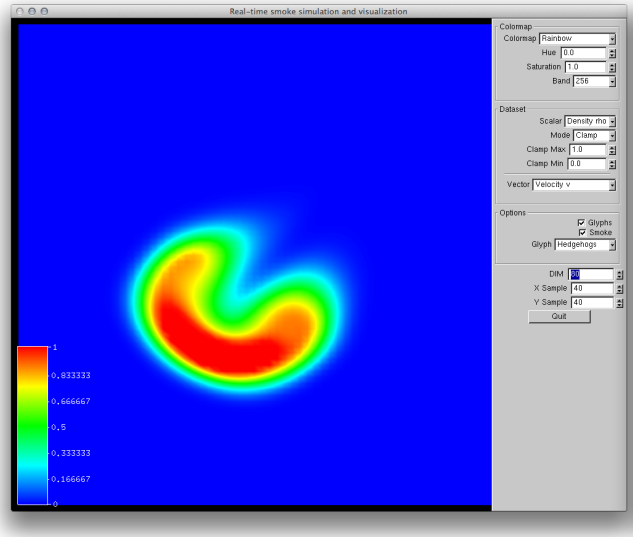

Figure 3: Fluid density visualized with a rainbow colormap.

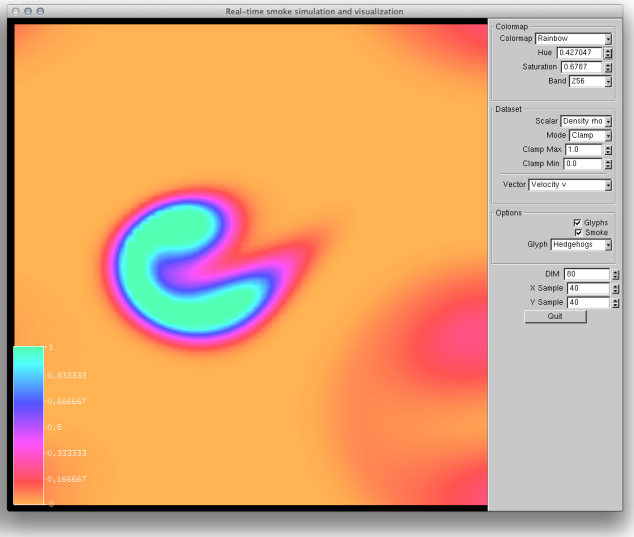

Figure 4: Fluid density with a less-saturated and hue-shifted rainbow colormap.

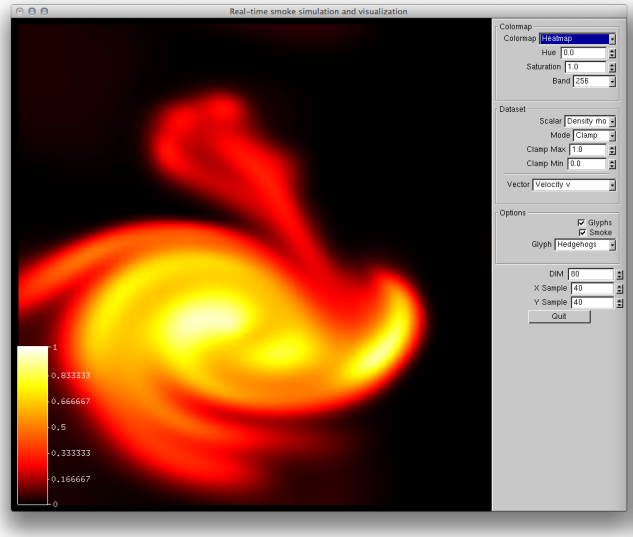

Figure 5: Fluid density visualized with a heat colormap

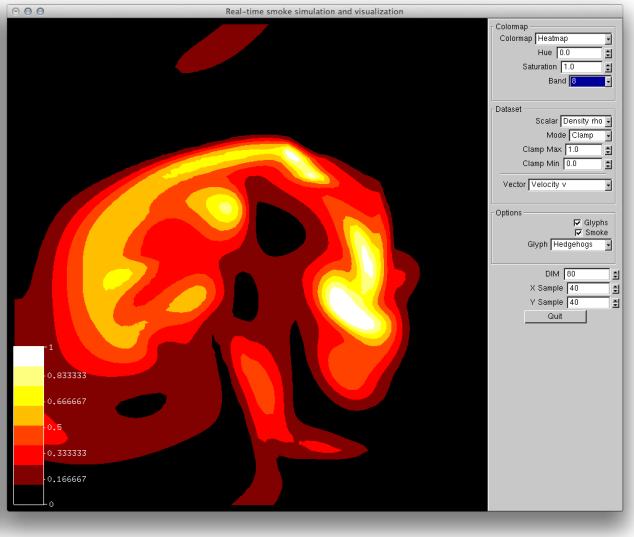

Figure 6: Heat colormap with a reduced number of colors. The color banding effect is clearly visible.

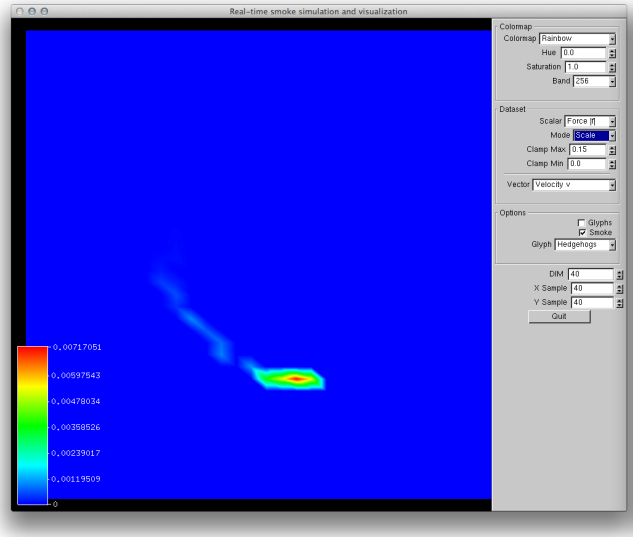

Figure 7: Scaling the colormap to the min and max of force always shows the maximum and minimum values at the current timestep although the values are quite small.

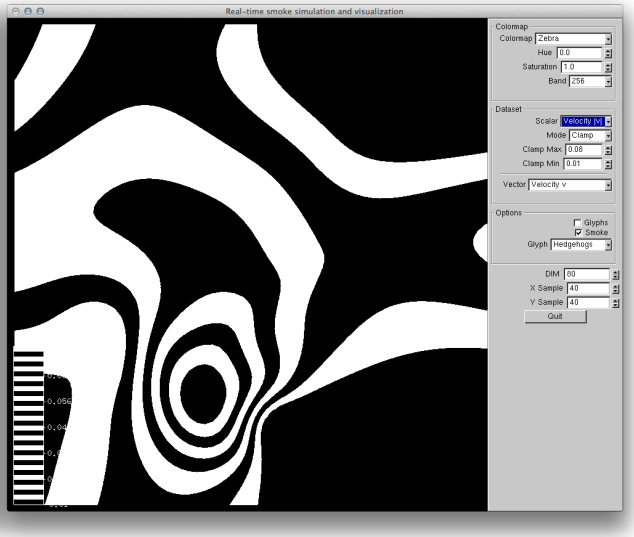

Figure 8: Velocity visualized with a zebra colormap that highlights areas with high variation.

2.3 Glyphs

The aim of this step is for you to understand and implement a set of glyph techniques. Glyphs are icons that convey the value (orientation, magnitude) of a vector field by means of several graphical icons, such as arrows. Glyphs can also be used to visualize several types of fields together, such as one scalar field and one vector field, at the same spatial locations. In this step, you will design and implement several glyphs for visualizing three datasets: The fluid density rho, the fluid velocity v, and the force field f. For a description of these quantities, see the skeleton code.

There is already a very basic implementation of arrow glyphs, also known as hedgehogs, provided in the skeleton code. Using this implementation as a starting point, and also the material in the book and lectures, you have to implement yourself several new functionalities, as follows.

· Implement a mechanism that lets you choose one scalar field and one vector field and visualizes their combination using glyphs. As scalar fields, you should be able to choose from: the density rho, the fluid velocity magnitude | v |, and the force field magnitude | f |. These are exactly the scalar fields used at step 2. As vector fields, you should be able to choose from: fluid velocity v and the force field f. Use the vector field direction and magnitude to control the orientation and length (respectively) of the arrows. Use the scalar field to control the arrows' colors. For this, use the color map techniques designed at step 2.

· Implement a mechanism to specify where to draw the glyphs. Start by modifying the mechanism currently implemented in the code which places the glyphs on a regular sampling of the grid. Provide interactive controls to specify the number of samples, in the x and y directions, where you want to evaluate the vector field. In case your sample points do not coincide with an actual computational grid point, provide interpolation mechanisms that compute the dataset values out of the grid neighbour point(s). Here, you can choose between nearest-neighbor (constant) and (bi)linear interpolation.

· Implement a mechanism that lets you parameterize the glyph itself. Besides using simple two-dimensional arrows (hedgehogs), implement two other glyph types that are able to show both a vector field and a scalar field. Suggestions include, but are not limited to:

three-dimensional cones three-dimensional ellipsoids three-dimensional arrows consisting of a cone tip and a cylindrical shafts arrows implemented as two-dimensional nice-looking, high-quality, textures instead of simple polygonal shapes

Just as for the previous steps, provide interactive means to choose the datasets and types of glyphs at runtime. Consider carefully the glyph design: How thick to make the glyph? How long to make it? Should you scale the length linearly with the vector magnitude, or use another scale? Should you clamp the glyph's minimal and maximal sizes to some values? If so, which are those?

2.4 Gradient

2.5 Streamlines

2.6 Slices

2.7 Stream surfaces

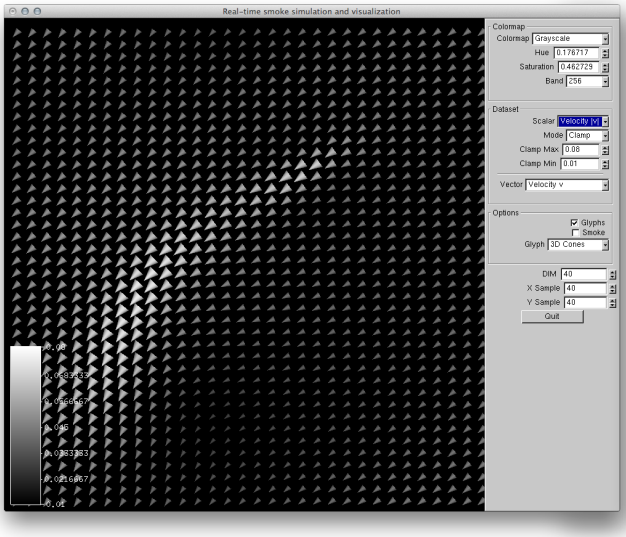

Figure 9

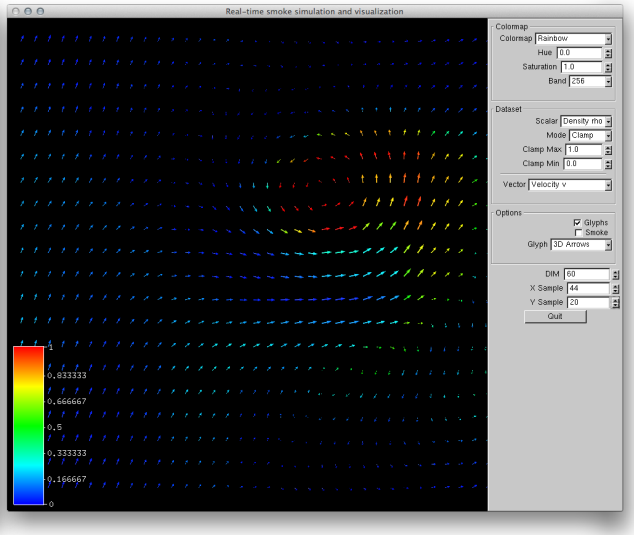

Figure 10

3 Conclusion

References