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

Color mapping 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 lookup table. These color points are stored in a map. The actual lookup-table will be generated by taking the color-points from the map and calculate the color by interpolating two adjacent color points. For instance blue at index 0, yellow at index 127 and green at index 255 will produce a colormap similar to the one shown in Figure 2(d).

Todo: the color points are stored in a map which is used to calculate the lookup table

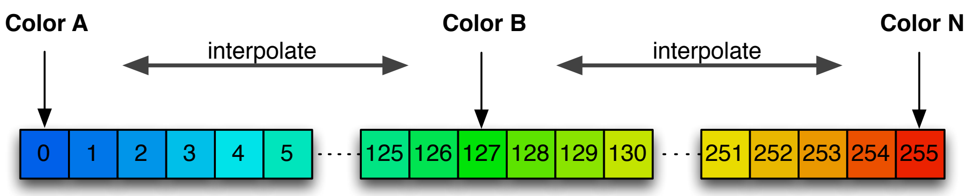


Figure 1: Conceptual model of the lookup-table. Missing colors are calculated by interpolating the defined color-points.

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 drag and 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.

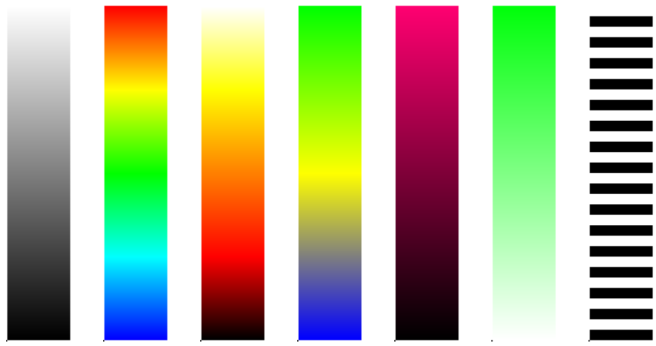
(d) **Blue-Yellow-Green** A colormap that separates the range of values into low, middle and high values.

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

(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.

(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.



(a) (b) (c) (d) (e) (f) (g)

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.

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

A less saturated and hue shifted version of the rainbow colormap is illustrated in Figure 4.

One is the fundamental problems is to map the values to a particular color, especially if the range of values is not known in advance or the dataset contains a few but extreme peaks or valley. Two different mapping modes have been implemented, which are clamping and scaling. Clamping allows the user to specify the minimum and maximum value, all values larger or smaller are clamped to the maximum respectively to the minimum value. In contrast, scaling uses the complete range of values in the dataset of the current timestep. This means that we have to calculate the current minimum and maximum value for every timestep before we map the values to a color.

The minimum and maximum value is necessary in order to translate the value range to the color range. This is done by scaling the value range up or down to the range of 0 to 255 by using the following code snippet. The resulting float value is rounded to its nearest integer value, which determines the index within the color table.

```
float scale(float v, float f_min, float f_max, float min, float max) {
    return ((v - f_min)*(max - min) / (f_max - f_min) + min);
}
```

Figure 9 depicts the visualization of the force magnitude by using a rainbow colormap and the scaling mode. The displayed colors extend over the complete range of available colors, although the values are very small and the range of values is quite narrow. During runtime it can be

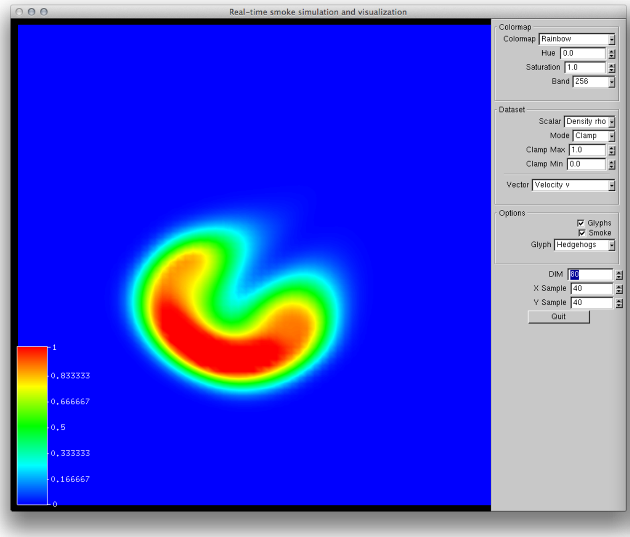


Figure 3: Fluid density visualized with a rainbow colormap.

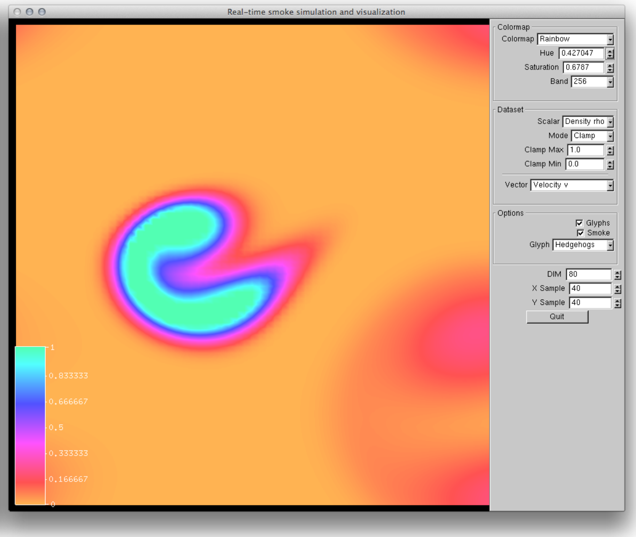


Figure 4: Fluid density with a less-saturated and hue-shifted rainbow colormap.

observed that the legend is constantly updated as the values become increasingly smaller unless new force is added.

In order to increase the accuracy we have implemented texture-based color mapping rather than vertex-based color mapping. The advantage is that not the color is interpolated between sample points but instead the index of the color. This produces better results as the mapped color is always part of the colormap. This is not automatically the case for vertex-based color mapping and thus requires a more careful design of the colormap itself. In order to use texture-based mapping we need to generate a texture whenever the colormap is changed (e.g. different colormap, hue, saturation, number of colors). The colormap is generated based on the lookup-table.

Another parameter that the user can influence is the number of colors, the less colors are specified in a colormap the more banding artifacts will be visible (cf. Figure 8, 7). We implemented banding by reducing the number of different colors in the lookup-table. The size of the lookup table itself always stays the same, which is 256 cells. This has the advantage that we do not need adapt the code for scaling or generating textures. As an example, we want to reduce the number of colors to 32. In concrete, this means the every eighth color in the base lookup-table will be spread over the following eight cells. The code is illustrated in Listing 1

Listing 1: Reducing the number of colors in the lookup-table

```
int step = 256 / numberOfColors;
 for (size_t i = 0; i < 256; i++) {

     int c = step * (int) floor(i / step);
     if (i >= 128) {
          c = c + step - 1;
     }
      colors[i] = colors[c];
 }
```

One of the most important aspects of visualization is the invert mapping of colors to data values in order to generate insight. A color legend allows to associate concrete colors with values by

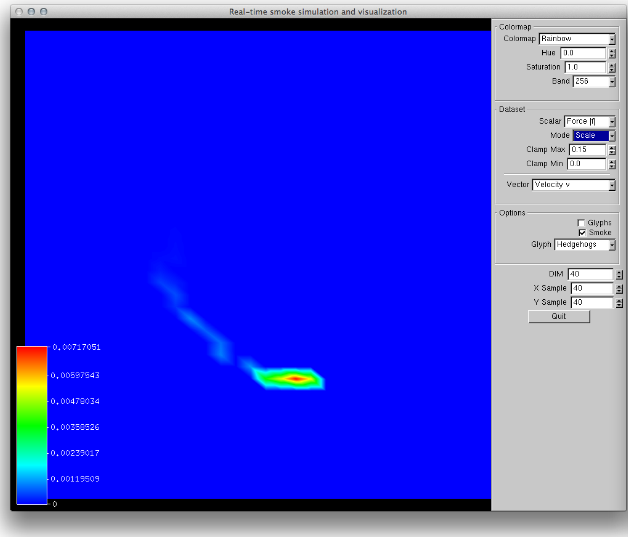


Figure 5: Scaling the colormap to the min and max of force always shows the maximum and minimum values at the current timestep although the values are quite small.

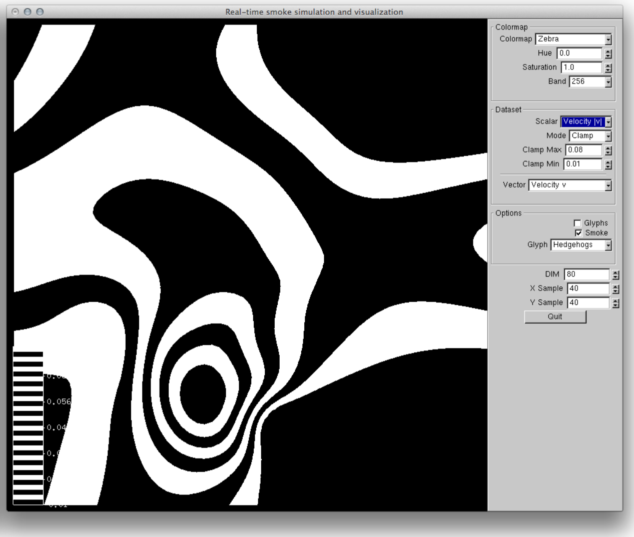


Figure 6: Velocity visualized with a zebra colormap that highlights areas with high variation.

showing both the colormap itself and the corresponding numerical values as shown in Figure 7.. However, the invert mapping is only possible with limited accuracy because it is not feasible to display all numerical values. The color legend is based on the lookup-table and the current minimum and maximum values of the dataset. Based on this information, the major and minor tick marks are calculated.

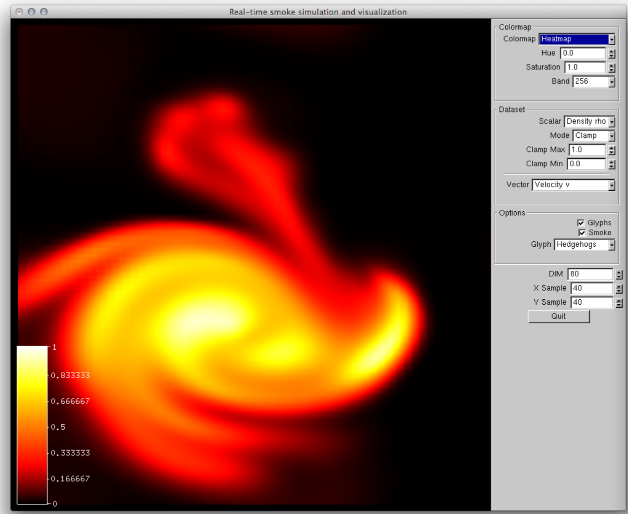


Figure 7: Fluid density visualized with a heat colormap

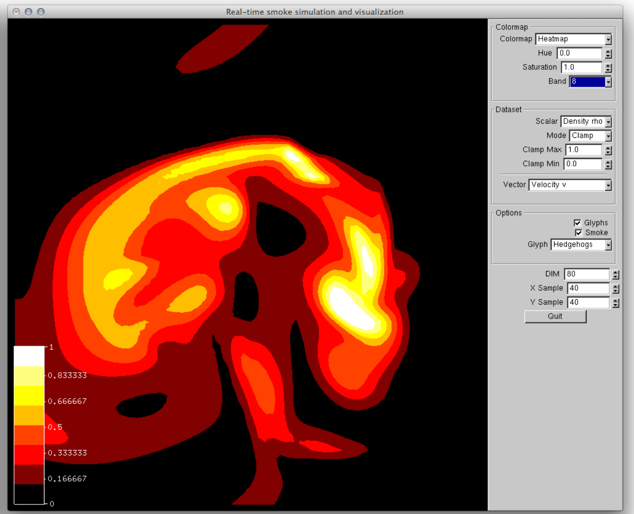


Figure 8: Heat colormap with a reduced number of colors. The color banding effect is clearly visible.

2.3 Glyphs

The dataset class, separation of concerns, map that stores vector and scalar dataset, max and min for colormap

changing the viewport to 3D

sub and supersampling vector interpolation (cf. exam question)

design of glyphs, factor, orientation hedgehogs 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 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?

combining colormapping and glyphs difficulties with colorizing glut objects

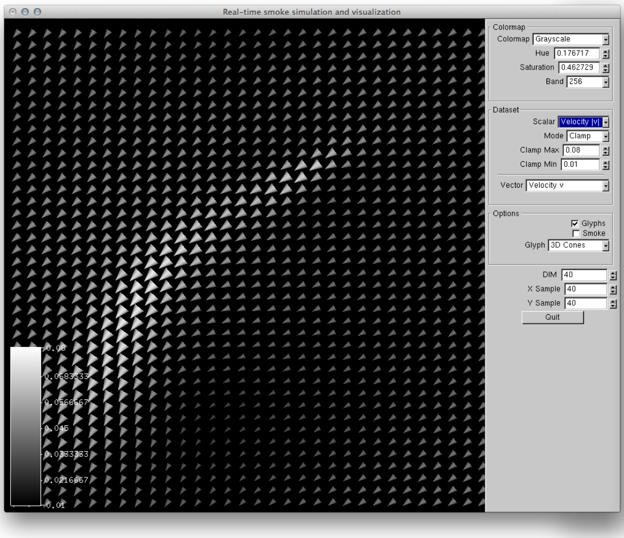


Figure 9

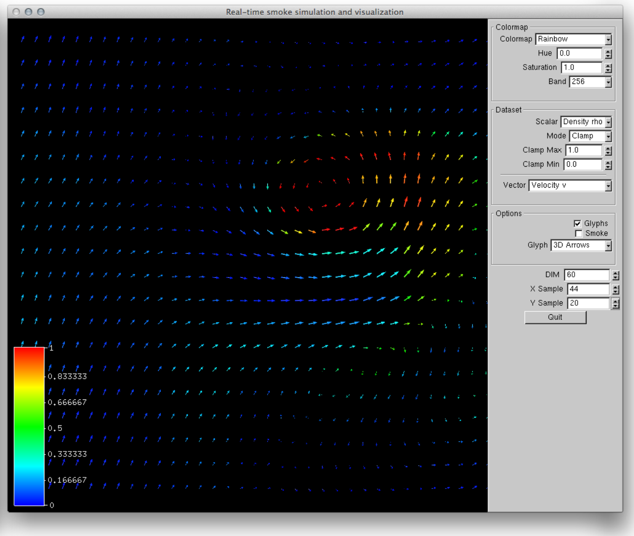


Figure 10

2.4 Gradient

2.5 Streamlines

2.6 Slices

2.7 Stream surfaces

2.8 User Interface

show final user interface

3 Conclusion

References