Final report

Christian Luijten

March 22, 2007

VÖKVAR, m.pl. liquids, fluids

Christian Luijten
Visualization (2IV30)
Department of Mathematics & Computer Science
Technische Universiteit Eindhoven

1 Introduction

This document describes the design process and usage of the VÖKVAR application as result of assignment 6 of the course of Visualization (2IV30) at the Technical University of Eindhoven.

The text of the assignment:

> In this assignment, you must construct a software tool for visualizing a time dependent two dimensional vector field that represents the flow of a fluid. This vector field is produced by a numerical simulation library called FFTW. Along with the flow field, FFTW simulates the diffusion and advection of matter in time. For an easy start, we provide a simple program that uses the FFTW to simulate and visualize a fluid flow. Our visualization is limited to simple hedgehogs colored by a blue-to-red colormap (see Course Material for details on these two algorithms). The simulation of the flow is driven interactively by the user, via mouse clicks in the window. The clicks inject local changes (vortices, matter) in the flow. The above software, written in C, can be downloaded from: `http://www.win.tue.nl/~alext/COURSES/INFO_VIS/SOFTWARE/Smoke.zip`.
>
> The assignment asks you to implement at least two of the following visualization techniques:
>
> - Streamlines: show the flow by a number of instantaneous streamlines. You should decide yourself about where to start (and stop) the streamlines, how long the streamlines should be, how many to draw, how to draw them, etc. Some of these parameters may be exposed to the end user via a user interface. Others may be automatically set by the program itself.
> - Isolines: show the flow by displaying the isolines of the matter density. You should decide upon the number of isolines, distance between isolines, and way of drawing them, just as in the case of streamlines.
> - Height plots: show the flow by displaying a height plot (3D elevation graph) of the matter density. You should decide upon the viewing angle, height mapping, shading, and coloring of the 3D plot.
>
> For this assignment, students can work in groups of two persons. The deliverable should contain:
>
> - a report containing a description of the implemented methods, the choices made during the implementation, the problems found, limitations of the proposed solution, and a simple 'user manual' of the implemented software, in total 5–10 pages.

- the software: source code as well as running executable.

Since the software we provide as a starting point is written in C under MS Windows (with Visual C++), we recommend you to develop your solution in the above setting. However, other implementations (Delphi, Java, Linux) are allowed too, as long as all deliverables are present. The deadline of this assignment is as for the other ones (before the spring term).

It was decided to implement streamlines and isolines, and to write the software in C under Mac OS X and Linux.

The final code compiles under Mac OS X and Linux and should in theory also compile under Windows given the correct build options.

1.1 The name: Vökvar

The author is fascinated by Iceland, the language and culture and recentrly lived in Iceland for four months during a traineeship. Vökvar is the icelandic word for fluids (vökvi being the singular).

2 Problems

This section identifies the problems to be solved when implementing streamlines and isolines in VÖKVAR.

2.1 Streamlines

To draw a perfect streamline, one has to calculate an integral for an absent function. In other words, this is impossible. The streamline can be approximated however by use of numerical integration.

There are various methods for doing this, Euler's and Runge-Kutta's techniques being the most famous. While Euler's method is known to be not very accurate when compared to Runge-Kutta's, the choice was put on the former.

2.2 Isolines

Isolines have the problem that it is impossible to determine the isovalues so that the whole window is evenly filled with lines.

There are three ways of setting those values:

- By fixed value, simply a list of predefined "clever" values, independent of the dataset.
- By fixed number, taking the current maximal and minimal value of the dataset, dividing it into a set number of isovalues.
- By fixed points, taking a point in the dataset and determining the isoline running through it.

All three methods are implemented using a simple version of the CONREC algorithm by Paul Bourke which was published in the Byte magazine issue of July 1987[1].

[1] `http://local.wasp.uwa.edu.au/~pbourke/papers/conrec/`

3 Design

The VÖKVAR component design is pretty straightforward. There are two main components; the simulation and the visualization.

3.1 Simulation

The basic functionality of this part is given by the `fluids-example.c` file. A data structure called Simulation is created to hold the information together. The listings in this section are taken from the file `simulation.h`.

```
/*********************************************************************
 * Public datastructures
 *********************************************************************/

typedef struct {
  int dimension; /* Size of the data set */

  fftw_real *u, *v; /* (u,v) = velocity field */
  fftw_real *u0, *v0;
  fftw_real *u_u0, *u_v0; /* User-induced forces */
  fftw_real *rho, *rho0; /* Smoke density */

  rfftwnd_plan plan_rc, plan_cr; /* FFT plans */
} Simulation;

typedef struct {
  float max; /* maximal value in the simulation data set */
  float min; /* minimal value in the simulation data set */
  float mean; /* mean value in the simulation data set */
} Simulation_statistics;

/*********************************************************************
 * Public functions
 *********************************************************************/

/* Creates a new simulation */
Simulation *new_simulation(int dimension);
/* Delete the simulation */
void simulation_destroy(Simulation *s);

/* Set the simulation forces */
void simulation_set_forces(Simulation *s);
/* Calculates new values */
void simulation_stable_solve(Simulation *s,
```

```
      fftw_real viscosity, fftw_real dt);
/* Lets matter diffuse */
void simulation_diffuse_matter(Simulation *s, fftw_real dt);
/* Returns the interpolated speed */
Vector *simulation_interpolate_speed(Simulation *s, Vector *v);
/* Returns the interpolated density value */
float *simulation_interpolate_density(Simulation *s, Vector *v);
/* Returns the density value at specified location */
float simulation_value(Simulation *s, int x, int y);
/* Returns the maximal density value in the data set */
float simulation_maximal_value(Simulation *s);
/* Returns a statistical information about the data set */
Simulation_statistics *simulation_statistics(Simulation *s);
```

3.2 Visualization

```
/**********************************************************************
 * Public datastructures
 **********************************************************************/

/* Visualization type selector */
typedef enum {
  VIZ_NONE = 0,
  VIZ_SMOKE = 1,
  VIZ_VECTORS = 2,
  VIZ_STREAMLINES = 4,
  VIZ_ISOLINES = 8
} Visualization_draw;

/* Isoline type selector */
typedef enum {
  VIZ_ISO_BY_NUM = 0,
  VIZ_ISO_BY_VALUE,
  VIZ_ISO_BY_POINT,
  VIZ_ISO_COUNT
} Visualization_isolines_type;

/* Visualization datastructure */
typedef struct {
  int width, oldwidth; /* Window width */
  int height, oldheight; /* Window height */

  int frozen; /* Simulation and visualization frozen? */
  int fullscreen; /* Window fullscreen? */
  int main_window; /* GL window identifier */
```

```
    float viscosity; /* Fluid viscosity value */
    float timestep; /* Time assumed between two calculaions */

    Visualization_draw draw;
    Vector *ratio; /* Ratio between data set coordinates and GL coordinates */

    int scalar_coloring; /* Palette selection */

    int color_dir; /* Color depending on direction of vectors? */
    float vector_scale; /* Scale factor for vectors vis. */

    Visualization_isolines_type isolines_type;
    void *isolines_datapoints; /* Storage for isolines module */
    int isolines_number; /* Number of isolines to draw */

    Simulation *simulation;
} Visualization;


/**********************************************************************
 * Public functions
 **********************************************************************/

/* Creates a new visualization */
Visualization *new_visualization(int argc, char **argv,
      Simulation *simulation, int width, int height);
/* Deletes a visualization */
void visualization_destroy(Visualization *v);

/* Starts the visualization */
void visualization_start(Visualization *v);
/* Stops the visualization */
void visualization_stop(Visualization *v);

/* Draws one frame in the visualization */
void visualization_draw_field(Visualization *v);
/* Sets the current color to the corresponding value in the palette currently
 * in use */
void visualization_set_color_palette(Visualization *v, float value);
```

3.2.1 Smoke

This was an existing visualization in the example and is integrated into VÖKVAR.

```
void smoke_draw(Visualization *v);
```

3.2.2 Vectors

This was an existing visualization in the example and is integrated into VÖKVAR.

void vectors_draw(Visualization *v);

3.2.3 Streamlines

```
6 void streamlines_draw(Visualization *v);
```

A streamline is generated by taking a point in the space and applying a function to it to get the next point it will be after some set time step.

VÖKVAR uses Euler's method, which means the function is of the form:

$$\overline{x}_{t+\Delta} = \overline{x}_t + \overline{v}_{x_t} \times \Delta$$

where $\overline{x}_t$ is the location of the particle to be traced at time t, $\overline{v}_{\overline{x}_t}$ is the velocity of that particle at that moment. Δ is the timestep size.

3.2.4 Isolines

The three methods of drawing imply three functions to call.

```
void isolines_draw(Visualization *v);
void isolines_draw_by_number(Visualization *v, int num);
void isolines_draw_by_value(Visualization *v, float *values, int num);
9 void isolines_draw_by_point(Visualization *v, Vector **points, int num);
```

The CONREC algorithm is explained extensively in the paper by Paul Bourke. It uses the observation that in a triangle, an isoline can only cut the triangle in two if there is at least 1 point under the isoplane and at least one above it.

Then by interpolation, a line is drawn over the triangle. Doing this with every triangle of adjacent data points, the whole plane is filled and thus isolines are constructed.

4 Usage

Upon startup, VÖKVAR is a black, square window. Use the mouse to click anywhere in the window and drag a little. Hereby, you are injecting fluid which will distribute and dissolve in the simulation.

To control the visualization, the keyboard is used. These are the commands:

- 1: Switches the "smoke" on and off
- 2: Switches the speed vectors on and off
- 3: Switches the streamlines on and off
- 4: Switches the isolines on and off
- p: Switches between palette (rainbow, less coloured rainbow, grayscales)
- o: Switches between isolines method (by value, by number, by point)
- a: Temporarily freeze simulation and animation
- f: Switches to and from fullscreen view
- t/T: Changes timestep
- s/S: Changes scale of vectors and streamlines
- v/V: Changes viscosity
- i/I: Changes number of isolines
- q: Quits VÖKVAR

4.1 Compilation

If you want to compile VÖKVAR yourself, you need to have the OpenGL and FFTW[2] libraries and headers installed.

In Debian, this is as simple as running the following the command:

```
# aptitude install fftw3-dev freeglut3-dev
```

In Mac OS X, fftw is available from Fink[3].

```
# fink install fftw3
```

The OpenGL libraries should be installed with the system (otherwise most of the graphical user interface wouldn't work).

After you installed the libraries, under Linux run `make -f Makefile.linux`, under Mac OS X run `make -f Makefile.macosx`.

There should be a binary called `vokvar`, run it.

[2] `http://www.fftw.org/`
[3] `http://fink.sourceforge.net/`

5 Conclusion

A working software tool was implemented, with streamlines and isolines. It was tested to compile and run properly on both Debian GNU/Linux and Mac OS X.

The initial source code needed some cleanup first, when it was clear how it worked, the design process started and resulted in the separate modules as they are now. The example code wasn't very clean and extensible, so this caused some head breakage at first. Especially because of unfamiliarity with OpenGL and FFTW.

5.1 Limitations

Some ideas that have come up during development, but which were dismissed because of scope or time concerns.

Two visualizations cannot both have a different palettes at the same time. This would be perfectly possible to implement, but usually one only uses one visualization at a time, so this won't be a big issue.

Free placement of streamlines would have been cool, would however also create the need for an interface to the OpenGL drag handler to react on mouse events.