Development of a simple numerical simulator of non-inertial fluids: 3D visualization in OpenGL Core Profile and multi/many-core parallelization in OpenCL

Donato D'Ambrosio

Course of Computer Graphics and GPGPU Programming
Master Degree in Computer Science
Department of Mathematics and Computer Science
University of Calabria, Italy

November 4, 2019

1 Introduction

The project consists in the development of a simple scientific application for the simulation of a non inertial fluid flow, like water, over a complex topographic surface.

The starting point is the *flow* application, which is a serial implementation of a simple fluid-dynamical simulator based on the Celluar Automata computational model [2], and the Minimization Algorithm of the Differences [1], this latter considered for computing flows among neighboring cells.

The input to the model is represented by two text files in Ascii Grid format. The considered data format is made by a header and a data section. The header provides information about domain dimensions (number of columns and rows), geographical coordinates (of the bottom left cell of the grid), cell size, and a no-data value (used to label the cells with a lack of information), while the data section, which is a regurlar grid (i.e., a matrix) of numerical values, represents the information associated to the (square) domain cells (for instance topographic elevation or mass thickness). An example is shown below and graphically represented in Figure 1:

```
ncols         10
nrows         10
xllcorner     2487289.5023187
yllcorner     4519797.0771414
cellsize      10.0
NODATA_value  -9999
9.00 8.00 7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00
8.00 7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00 1.00
7.00 6.00 5.00 4.00 3.00 2.00 1.00 0.00 1.00 2.00
6.00 5.00 4.00 3.00 2.00 1.00 0.00 1.00 2.00 3.00
5.00 4.00 3.00 2.00 1.00 0.00 1.00 2.00 3.00 4.00
4.00 3.00 2.00 1.00 0.00 1.00 2.00 3.00 4.00 5.00
3.00 2.00 1.00 0.00 1.00 2.00 3.00 4.00 5.00 6.00
2.00 1.00 0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00
1.00 0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00
0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00 9.00
```

Each cell of the square grid interacts with a small set of neighboring cells adjacent to it. In this model, the considered stencil is the von Neumann one that, besides the central cell, takes into account the adjacent cells along the North, Est, West and South directions. An index is assigned to each cell: 0 to the central one, 1 to the north neighbor, and so on, as shown below.

```
           |1:(-1, 0)|
 |2:( 0,-1)|0:( 0, 0)|3:( 0, 1)|
           |4:( 1, 0)|
```

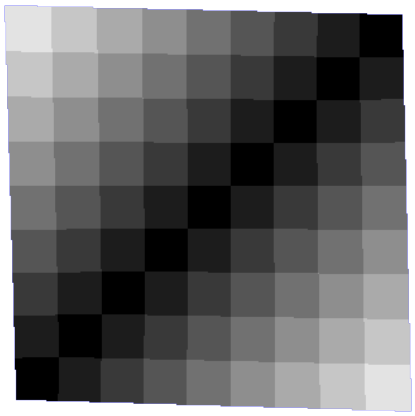


FIGURE 1: Graphical representation of the 10x10 Ascii Grid. It is based on a straightforward color mapping technique, which simply assigns the color black to the cell with the lowest value, white to the one with the highest value and grey tones to cells having intermediate values.

Cells IDs (from 0 to 4) and relative (row, column) coordinates with respect to the central cell (having relative coordinates (0,0)) are also shown in the scheme reported above.

The cell state is defined by the following information (substates):

- topographic altitude, z;
- mass thickness, h;
- values of the four outflows from the central cell towards its neighboring cells, $o[i]$ $(i = 1, ..., 4)$.

Each substate layer, which formally is a 2D matrix of real values, is coded as a linearized array of type double*, arranged by rows. In particular, the *flow* application considers the following layers:

- `Sz` is the layer representing the z information;
- `Sh` is the layer representing the h information;
- `So[0]` is a fictitious layer (memory is not allocated) used to let the other flow indices to correspond to the proper neighboring cell (since the central cell has index 0);
- $So[1]$ is the layer representing the amount of h that otflows towards the neighbor with index 1
- $So[2]$ is the layer representing the amount of h that otflows towards the neighbor with index 2

- $So[3]$ is the layer representing the amount of h that otflows towards the neighbor with index 3
- $So[4]$ is the layer representing the amount of h that otflows towards the neighbor with index 4

Two macros (`get` and `set`) are used to access values defining the cell's state (substates) based on the cell's integer coordinates. Please, note that the number of columns must also be provided to the macros, since the linarized matrices are arrangeb by rows (and therefore the number of columns represents the number of elements in each row).

The initial configuration of the system is defined as follows:

- a topographic map is read from a file to initialize the `Sz` substate layer;
- a mass thickness map is read from a file to initialize the Sh substate layer;
- the outflow substate layers are initilized to zero.

Boundary conditions are very simplyfied and consist in ignoring (i.e. not evaluating the state transition for) the cells belonging to the grid boundaries.

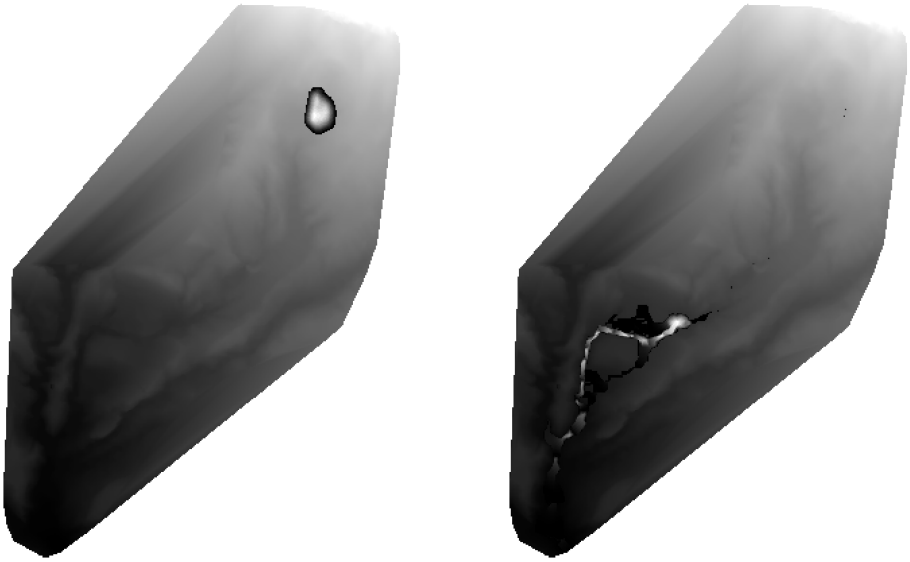
At each iteration step, three basic sub-steps (aka kernels, local processes, or elementary processes) are computed:

- `kernel1: flow computation`. Computes the outflows from the central cell to the neigboring cells (by applying the Minimization Algorithm of the Differences). The algorithm takes in intput the total head (H=z+h) of the cells belonging to the neighborhood and evaluates the outflows by minimizing unbalance conditions. Computed outflows are therefore stored into the outflow substate layers So[i] (i=1,...,4).
- `kernel2: mass balance`. New thickness values are evaluated by subtracting the outgoing flows from the central cell and by adding the ingoing flows coming from the adjacent neighbors. The `Sh` layer is updated.
- `kernel3: outflow`. The `So[i] (i=1,...,4)` layers are set to zero.

To build the progrma, open a terminal window, go to the application directory (the one containing the CMakeLists.txt file) and give the following command:

```
cd build/ && cmake -DCMAKE_BUILD_TYPE=Release ../ && make && cd ..
```

Eventually, give the command below to perform a simulation of 4000 steps using the `./data/dem.asc` and and `./data/source.asc` files to initilize the `Sz` and the `Sh` substate layers, respectively:



(A) Initial configuration (B) Final configuration

FIGURE 2: Initial and final configuration (after 4000 computational steps) of the system.

```
./build/flow ./data/dem.asc ./data/source.asc 4000
```

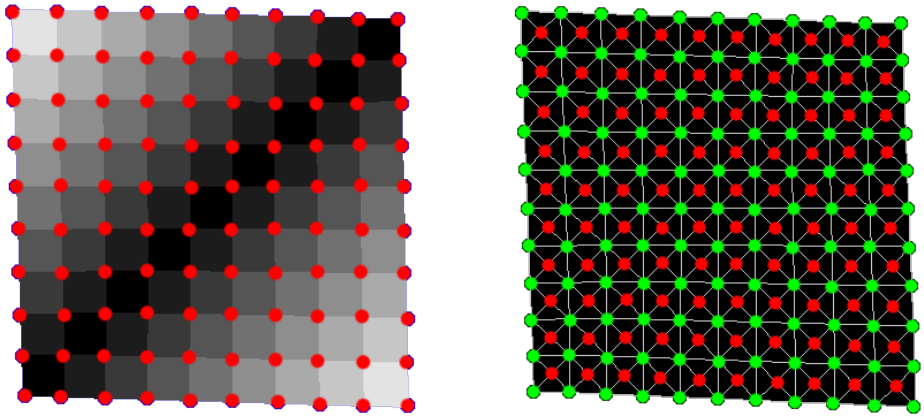
The (serial) simulation should last less than half a minute and, as a result, the `./data/source.asc.out` file will be created, with the final configuration of the `Sh` substate layer. Figures 2a and 2b show the initial and final configuration of the system, respectively.

2 Graphics

A GLFW application must be developed based on the `flow` application in order to render the simulation at prefixed steps (e.g. at multiples of 100 steps). In order to graphically render the system configuration (`Sz` and `Sh` layers), follow the steps described below.

As a first preliminary step, simply represent the topographic surface by using triangle strips, a triangle strip for each row of the matrix. We will refer such a representation as grid representation. The color of the resulting cells has to be assigned by considering the value of the upper left point. In this way, a matrix of $n \cdot m$ points produces $(n-1) \cdot (m-1)$ cells, as illustrated in Figure 3a. Pay attention to the proper management of `NODATA_value`, that should not be displayed.

As a second step, draw the surface representing a cell for each point in the matrix. The cell must be built around the known point, as shown



(A) Initial configuration (B) Final configuration

FIGURE 3: (A) Grid-based representation of the example surface, where known altitudes are evidenced by red points. (B) Cell-based representation of the same surface (known altitudes are still evidenced in red). A cell, made by 4 further vertices (in green), is built around each known point.

in Figure 3b. In this way the color of the cell can be assigned by simply considering the elevation of the cell itself. Note that, using the cell-based representation, a matrix of $n \cdot m$ points produces exactly $n \cdot m$ cells. Again, pay attention to the proper management of `NODATA_value`, that should not be displayed.

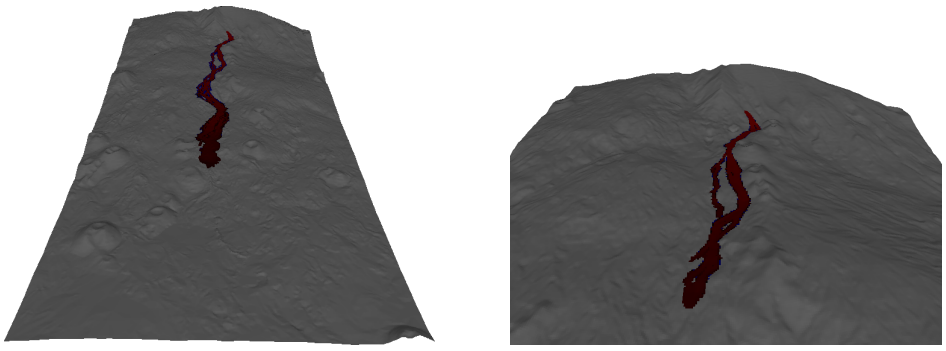
Once the topographic data, i.e. `Sz`, has been properly represented, the mass thickness, `Sh`, can be simply rendered by considering the same procedure, adopting a different color legend (for instance a green or reed one) to represent tickness values. An example is reported in Figure 4b, where both a 3D representation of the system is shown and the Phong lighing model is considered for shading the different domain cells depending on their orientation into the 3D space.

3 Parallel Computing

Under construction...

References

[1] S. D. Gregorio, R. Serra, An Empirical Method for Modelling and Simulating some Complex Macroscopic Phenomena by Cellular Automata, Future Generation Computer Systems 16 (1999) 259–271.



(A) Simulation 3D view (B) Detail of 3D view

FIGURE 4: Three dimensional representation of the system. The topographic layer is represented, together with the mass thickness. The Phong model is usd to shade the topographic data, based on the cells orientation, while a direct red-tones color mapping is used to shade the mass thickness.

[2] J. von Neumann, Theory of Self-Reproducing Automata, University of Illinois Press, Champaign, IL, USA, 1966, edited by Arthur W. Burks.