Real-Time 2.5D Fluid Dynamics Simulation Using GPGPU

Žiga Leber *
Supervised by: Niko Lukač †

Laboratory for Geometric Modeling and Multimedia Algorithms
University of Maribor, Faculty of Electrical Engineering and Computer Science
Smetanova ulica 17, SI-2000 Maribor, Slovenia

Abstract

Realistic depiction of fluids has always presented a challenge in the modern entertainment industry, partially owing to the fact that the Navier-Stokes equations for fluid dynamics are notorious for being highly computationally intensive. As a solution, a simple approximative 2.5D fluid dynamics model is proposed based on Shallow Water Equations and Continuous Cellular Automata. In combination with the computational abilities of the modern CUDA-enabled graphics cards, the simulation with the proposed model coupled with rendering, can be performed in real-time. The method was tested on high-resolution digital terrain models, where visually convincing results were achieved; additionally, the simulation scales well with the input size. The CUDA-based approach achieved average speedups of 5.5 and 1.7 times in comparison to the single and multi-core CPU implementations, respectively.

Keywords: Fluid dynamics, Shallow Water Equations, Continuous Cellular Automata, CUDA, GPGPU

1 Introduction

The audience has grown accustomed to perpetual increase of realism in computer graphics, which has put the entertainment industry under increasing pressure to depict accurately the phenomena that have been traditionally neglected due to high complexity. Fluid flow, with its ever-changing irregular shapes and computationally intensive methods of simulation, is definitely one of those. Consequently, this has sparked an increased interest in physically-based fluid simulation.

Fluid flow is governed by a system of equations which describe its three fundamental properties:

1. Conservation of mass,
2. Conservation of momentum,
3. Conservation of energy.

The formulation of equations depends on the frame of reference that was selected upon derivation [14].

Following the Lagrangian approach, the fluid is modelled as a particle system. The particles are described in terms of positions and velocities. The method simplifies interactions with irregular boundaries and between multiple fluids. However there are difficulties with the surface reconstruction and rendering. Although particle-based methods cannot achieve the same level of detail as Eulerian, they are computationally less expensive. A popular method used for simulation of such systems is Smoothed Particle Hydrodynamics (SPH) [14]. It is defined with a smoothing kernel used for interpolating the physical properties of the fluid from neighbouring particles. This approach was used in the work of [12], where they used SPH to model the Shallow Water Equations (SWE) on arbitrary terrain slopes. Additionally, the surface of the fluid can be rendered in real-time [14].

The Eulerian methods model the fluid as a grid and follow the flow inside the cells. The state is stored inside the node of each cell. The more advanced configurations use staggered grids [6], where the velocity is stored instead at the grid cells' faces. The equations can be solved using different methods; the most common are the finite difference, finite elements and finite volume methods [14]. A pivotal work in this area was done by Stam et al. [13], which first introduced an unconditionally stable numerical solver. It is based on a semi-Lagrangian method for advection and implicit method for viscosity and pressure terms. However, the method cannot model fluids with free boundaries [13]. In [4] Dong et al. used an adaptive SWE solver which regulates the level of detail based on the distance to the viewpoint and the velocity gradient. The technique enables dynamic grid refinement of areas that are of interest to the viewer, thus increasing the perceived resolution without raising the computational costs.

Another option is to model the fluid based on its microscopic properties as is done with the Lattice Boltzmann Method (LBM). The method is a simple kinetic model governed by the Boltzmann equation, which, when averaged, obeys the microscopic properties of the Navier-Stokes equation. The method is at an advantage over conventional methods when dealing with complex boundaries, microscopic interactions such as multiphase flows and wetting on solid surfaces. Additionally, it can be parallelized easily [14].

The SWE are derived by depth integrating the full Navier-Stokes equations. This simplifies the mathematical formulation of the equations vastly, since the pressure distribution can be assumed to be hydrostatic. The

*ziga.leber@um.si
†niko.lukac@um.si

limitation of the formulation is that, for the equations to hold, the horizontal scale of the flow must be much larger than the vertical dimension [16, 2]. The simplification turns the equations from 3D to 2.5D, which lowers the time and space complexity of the simulation, which is a good compromise if the limitations are acceptable. Nonetheless, there are various phenomena that the equations are capable of simulating without much loss of detail. The equations are especially suitable for simulating flows of surface water, such as rivers, tides, dam breaks and tsunamis. In spite of the name, they can also be used to simulate flows unrelated to water, they have been applied successfully in simulation of the atmosphere in weather forecasting and landslide models [16].

In this paper, an Eulerian model was developed for physically-based simulation of incompressible fluid flows based on SWE solved using a Finite Volume Method (FVM) in combination with the local Lax-Friedrich discretization. The wetting/drying problem was solved using a Continuous Cellular Automata (CCA). The method is simple to implement and easily parallelizable. It was implemented in three variants, for a single core CPU, a multi-threaded CPU, and for the Compute Unified Device Architecture (CUDA) platform on the GPU. All implementations were tested on different problems and the results compared. There was a substantial speedup using the GPU implementation, which shows the method scales well with added computational resources.

This paper is organised as follows: In the next Section we present the used methodology, which is divided into a description of the used methods and the way they were parallelized on the GPU. In Section 3 we present the obtained results and we compare the performance on different hardware. The paper is concluded in Section 4.

2 Methodology

In this section, first follows the explanation of the theoretical background and presentation of both components, and then the specifics of the implementation on the GPU.

2.1 Theoretical Background

The proposed method for fluid simulation consists of a numerical solver of SWE and CCA. Both components work on a 2.5D grid, which means that the state of the computation is represented in the form of a discretized scalar field. Considering that each cell contains the average height in that region, there exists a one-to-one mapping to a surface in 3D space. The computational domain of the method extends over the entire grid. The model works by alternating both methods, first an iteration of the CCA is run, which is followed by the numerical solver, if the height of the free surface is above zero. If that is not the case the step is skipped. The process is repeated until convergence. The following subsections explain each of the components.

2.1.1 Shallow Water Equations

The 2D Shallow Water Equations [3, 15] can be written as the following system of partial differential equations:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{G}(\mathbf{U})}{\partial y} = \mathbf{S}, \tag{1}$$

in which $\mathbf{U}$ contains the conservative variables, $\mathbf{F}$ and $\mathbf{G}$ contain the flux terms in the vertical and the horizontal directions, respectively, and $\mathbf{S}$ is the source term.

$$\mathbf{U} = \begin{bmatrix} \eta \\ hu \\ hv \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}, \tag{2}$$

where h is the depth of the fluid, $\eta = h + z$ is the elevation of the free surface from a fixed reference point, and z is the elevation of the bottom boundary. The parameter g represents the acceleration due to gravity, u and v are the horizontal and vertical components of the depth averaged velocity. The source term represents the effects of the hill slope and the friction at the bottom boundary:

$$\mathbf{S} = \begin{bmatrix} 0 \\ g\eta S_{0x} - ghS_{fx} \\ g\eta S_{0y} - ghS_{fy} \end{bmatrix}, \tag{3}$$

where S_0 is the hill slope gradient defined as:

$$S_{0x} = -\frac{\partial z}{\partial x}, \quad S_{0y} = -\frac{\partial z}{\partial y}. \tag{4}$$

S_f is the friction defined by the Manning formula [15], parametrized by the roughness coefficient a:

$$S_{fx} = \frac{a^2 u\sqrt{u^2+v^2}}{h^{4/3}}, \quad S_{fy} = \frac{a^2 v\sqrt{u^2+v^2}}{h^{4/3}}. \tag{5}$$

The equations are solved using the FVM [11, 8]. Firstly, the problem domain is decomposed into subdomains, called Control Volumes (CV). As the proposed method is based on a 2.5D grid, the most natural decomposition is to assign a CV to each cell, with the nodes positioned at the centre (cell-oriented arrangement) [11]. The nodes represent the location where the unknown variables are calculated. Control volumes in this case are actually surfaces, owing to the fact that the governing equations are two-dimensional; however, the term volume is used because of the established nomenclature. Secondly, the integral balance equations have to be specified for each CV. The Shallow Water Equations, thus, have to be rewritten in the integral form:

$$\frac{\partial}{\partial t}\int_\Omega \mathbf{U}\,\mathrm{d}\Omega + \int_\Omega \left(\frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y}\right)\mathrm{d}\Omega = \int_\Omega \mathbf{S}\,\mathrm{d}\Omega, \tag{6}$$

where Ω represents the surface of the CV. Using Green's theorem, the surface integral of the flux can be written as the flux through the cell boundaries [3, 15]:

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{U}\,\mathrm{d}\Omega+\oint_{\partial\Omega}(\mathbf{F}\,\mathrm{d}y-\mathbf{G}\,\mathrm{d}x)=\int_{\Omega}\mathbf{S}\,\mathrm{d}\Omega. \quad (7)$$

Thirdly, the integrals have to be approximated numerically. Conservative and source terms are integrated using the midpoint rule, the flux is calculated separately for each edge of the cells and the time derivative is approximated using the forward Euler time-stepping scheme [8]:

$$\begin{aligned}\frac{\mathbf{U}^{n+1}_{i,j}-\mathbf{U}^{n}_{i,j}}{\Delta t}+\frac{1}{\Omega_{i,j}}(&\mathbf{F}^{n}_{i+\frac{1}{2},j}\Delta y-\mathbf{F}^{n}_{i-\frac{1}{2},j}\Delta y+\\&\mathbf{G}^{n}_{i,j+\frac{1}{2}}\Delta x-\mathbf{G}^{n}_{i,j-\frac{1}{2}}\Delta x)=\mathbf{S}_{i,j},\end{aligned} \quad (8)$$

where n is the number of the current iteration, i and j are the cell indices, $\Omega_{i,j}=\Delta x\Delta y$ is the area of the cell, and $\mathbf{U}_{i,j}$ are the averaged values of the conservative variables, inside the cell. The equation can be rewritten as:

$$\begin{aligned}\mathbf{U}^{n+1}_{i,j}=\mathbf{U}^{n}_{i,j}-&\frac{\Delta t}{\Delta x}(\mathbf{F}^{n}_{i+\frac{1}{2},j}-\mathbf{F}^{n}_{i-\frac{1}{2},j})-\\&\frac{\Delta t}{\Delta y}(\mathbf{G}^{n}_{i,j+\frac{1}{2}}-\mathbf{G}^{n}_{i,j-\frac{1}{2}})+\Delta t\mathbf{S}_{i,j}.\end{aligned} \quad (9)$$

The fluxes at the boundaries are calculated according to the local Lax-Friedrichs method [8]. Eq. (10) calculates the flux at the edges.

$$\begin{aligned}\mathbf{F}^{n}_{i+\frac{1}{2},j}&=\frac{1}{2}\left[\mathbf{F}(\mathbf{U}^{n}_{i,j})+\mathbf{F}(\mathbf{U}^{n}_{i+1,j})\right]-\frac{1}{2}|\lambda^{\vec{x}}_{i+\frac{1}{2},j}|(\mathbf{U}^{n}_{i+1,j}-\mathbf{U}^{n}_{i,j}),\\
\mathbf{F}^{n}_{i-\frac{1}{2},j}&=\frac{1}{2}\left[\mathbf{F}(\mathbf{U}^{n}_{i,j})+\mathbf{F}(\mathbf{U}^{n}_{i-1,j})\right]-\frac{1}{2}|\lambda^{\vec{x}}_{i-\frac{1}{2},j}|(\mathbf{U}^{n}_{i,j}-\mathbf{U}^{n}_{i-1,j}),\\
\mathbf{G}^{n}_{i,j+\frac{1}{2}}&=\frac{1}{2}\left[\mathbf{G}(\mathbf{U}^{n}_{i,j})+\mathbf{G}(\mathbf{U}^{n}_{i,j+1})\right]-\frac{1}{2}|\lambda^{\vec{y}}_{i,j+\frac{1}{2}}|(\mathbf{U}^{n}_{i,j+1}-\mathbf{U}^{n}_{i,j}),\\
\mathbf{G}^{n}_{i,j-\frac{1}{2}}&=\frac{1}{2}\left[\mathbf{G}(\mathbf{U}^{n}_{i,j})+\mathbf{G}(\mathbf{U}^{n}_{i,j-1})\right]-\frac{1}{2}|\lambda^{\vec{y}}_{i,j-\frac{1}{2}}|(\mathbf{U}^{n}_{i,j}-\mathbf{U}^{n}_{i,j-1}),\end{aligned} \quad (10)$$

where $|\lambda^{\vec{x}}|$ and $|\lambda^{\vec{y}}|$ represent the speed at the boundary, which is defined as the largest absolute eigenvalue of the Jacobian matrix. The Jacobian matrices are evaluated for the vertical and the horizontal flow [10] as:

$$\mathbf{A}=\frac{\partial\mathbf{F}}{\partial\mathbf{U}},\quad \mathbf{B}=\frac{\partial\mathbf{G}}{\partial\mathbf{U}}, \quad (11)$$

The calculation for the horizontal direction produces:

$$\mathbf{A}=\begin{bmatrix}0 & 1 & 0\\ gh-u^2 & 2u & 0\\ uv & v & u\end{bmatrix}, \quad (12)$$

which is decomposed, giving us the eigenvalues representing the characteristic speeds:

$$\lambda^{\vec{x}}_1=u+\sqrt{gh},\quad \lambda^{\vec{x}}_2=u-\sqrt{gh},\quad \lambda^{\vec{x}}_3=u. \quad (13)$$

The one with the largest absolute value in the horizontal direction is, therefore, $|\lambda^{\vec{x}}|=|u|+\sqrt{gh}$. The calculation in the vertical direction is similar and gives $|\lambda^{\vec{y}}|=|v|+\sqrt{gh}$. The average of the heights and the average of the velocities on each side are used to get the value of $|\lambda^{\vec{x}}|$ or $|\lambda^{\vec{y}}|$ at the boundary.

The time-step is determined dynamically according to the Courant-Friedrichs-Lewy (CFL) condition [8], which is necessary for the convergence of the numerical scheme:

$$\Delta t=c\cdot\min\left(\frac{\Delta x}{|\lambda^{\vec{x}}|},\frac{\Delta y}{|\lambda^{\vec{y}}|}\right), \quad (14)$$

where c is the Courant number. For this particular scheme c has to be at least smaller than 1 to avoid oscillations. No special treatment is given to the source term. The slope of the bottom surface is discretized using the central difference and the term is then calculated using the information provided by the 2.5D grid.

2.1.2 Continuous Cellular Automata

CCA is an extension of cellular automata, where the valid states can take continuous values. Generally, it is represented as a 4-tuple (Z,S,N,f), where Z is the lattice, S is a set of cell states, N is the finite neighbourhood, and f is the transition function. In this paper, the CCA model lattice is a finite 2.5D grid. The state is a real value that represents the height of the fluid. Each cell in the 2.5D grid adheres to the transition rules that define the fluid propagation from the given cell to the cells in the 8-neighbourhood. The movement of the fluid from the (i,j)-th cell in the r-th direction is governed by the slopes of the surrounding terrain. Therefore, the force $\mathbf{f}^{\vec{r}}_{i,j}$ applied to the fluid inside the cell is estimated with the Newtonian mechanics [7]:

$$\mathbf{f}^{\vec{r}}_{i,j}=m_{i,j}\mathbf{g}\cdot\sin(\phi^{\vec{r}}_{i,j}), \quad (15)$$

where $m_{i,j}$ is the mass of the fluid within the cell, $\mathbf{g}$ is the gravitational force, and $\phi^{\vec{r}}_{i,j}$ is the angle of the slope. The slope angles are calculated based on the heights of the neighbouring cells:

$$\phi^{\vec{r}}_{i,j}=\mathrm{atan}(\eta_{i,j}-\eta^{\vec{r}}_{i,j}). \quad (16)$$

The amount of fluid that is distributed to the neighbouring cells depends on the forces $\mathbf{f}^{\vec{r}}_{i,j}$ acting in each direction. The fraction of the distributed fluid $d^{\vec{r}}_{i,j}$ is, thus:

$$d^{\vec{r}}_{i,j}=\frac{m_{i,j}\mathbf{g}\cdot\sin(\phi^{\vec{r}}_{i,j})}{\sum_{k=0}^{8}\left(m_{i,j}\mathbf{g}\cdot\sin(\phi^{\vec{k}}_{i,j})\right)}=\frac{\sin(\phi^{\vec{r}}_{i,j})}{\sum_{k=0}^{8}\sin(\phi^{\vec{k}}_{i,j})}, \quad (17)$$

where the mass and the gravity cancel out. The total fluid that can be distributed in a given direction depends on the height of the neighbouring cells. The fluid cannot move uphill; therefore, only the fluid that is above the surrounding fluid surface can be moved. The height of the transferable fluid in each direction $\hat{\eta}^{\vec{r}}_{i,j}$ is given by:

$$\hat{h}^{\vec{r}}_{i,j}=\max(0,\eta_{i,j}-\eta^{\vec{r}}_{i,j}). \quad (18)$$

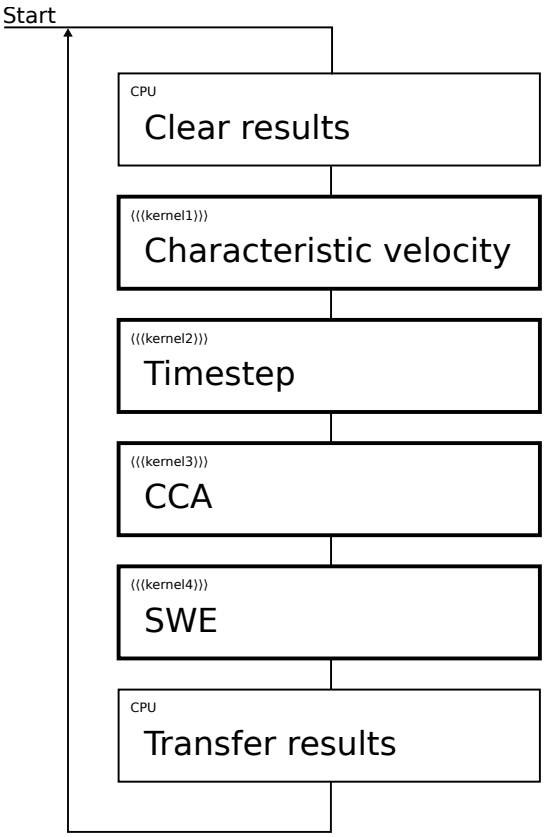

Figure 1: Workflow of the proposed CUDA-based implementation for parallel SWE+CCA computation.

In the case that the transferable fluid is negative, no fluid is transferred. The height of the fluid distributed in each direction $\Delta^{\vec{r}}_{i,j}$ is calculated as:

$$\Delta^{\vec{r}}_{i,j} = \min\left(\mu \hat{h}^{\vec{r}}_{i,j},\ h_{i,j}\frac{d^{\vec{r}}_{i,j}}{w^{\vec{r}}}\right), \tag{19}$$

where μ represents a damping factor, which limits the amount of the fluid that is distributed, $h_{i,j}$ is the height of the fluid at the current position, and $w^{\vec{r}}$ is the weight for the given direction. The weight is used because the fluid, which moved in a diagonal direction, has to cover more ground. The value of $w^{\vec{r}}$ for the orthogonal directions is 1 and $\sqrt{2}$ for the diagonal.

After the new heights are calculated using the CCA propagation rule in Eq. (19), the fluid propagates to the neighbouring cells. This process is repeated until convergence, when the height of the transferable fluid is negligible or the simulation is stopped by hand.

2.2 CUDA Implementation

CUDA is a parallel computing platform and a programming model which enables the use of GPU for general purpose computing. The CUDA enabled device is composed of two main components: The global memory which stores the data that is being processed and Streaming Multiprocessors (SM) that perform the computations. Each SM contains its own caches, shared memory, control units, execution pipelines and multiple CUDA cores which execute instructions in unison. The methods implementation is composed of serial code that is executed on a CPU and parallel code, composed of kernels, executed on different threads on a GPU. Threads are grouped into blocks, which are, in turn, grouped into a grid [9, 17].

The proposed CUDA implementation of the simulation uses four kernels as can be seen in Figure 1. Before running the simulation loop, the model of the terrain and the initial height are loaded into the global memory of the GPU. The velocities in both direction are initialized to zero. The constants are loaded into the fast read-only constant memory. In each iteration of the loop, the results are first set to zero, because the CCA uses these arrays as accumulators. When the first kernel is run, it computes the characteristic velocities and their maximum. Next, the time-step is calculated by using a small kernel which only executes once. Then follow the main parts of the simulation, where the third kernel executes a round of CCA and the fourth kernel applies the SWE solver.

Algorithm 1 CCA

```
1:  procedure KERNEL3(i, j, m, h, u, v, h', Θ, Π)
2:      if OUTOFBOUNDS(i, j) then
3:          return
4:      end if
5:      if h_ij < Θ or MAGNITUDE(u, v) < Π then
6:          h'_ij ← h'_ij + h_ij
7:          return
8:      end if
9:      slopes ← {0...0}
10:     ts ← 0
11:     for k ← 0 to 8 do
12:         r ← neighbourhood_k
13:         s ← SLOPE(i, j, r, m, h)
14:         if s ≥ 0 and h_ij > 0 then
15:             slopes_k ← s
16:             ts ← ts + s
17:         end if
18:     end for
19:     height_left = h_ij
20:     for k ← 0 to 8 do
21:         r_1, r_2 ← neighbourhood_k
22:         s ← slopes_k
23:         th ← TRANSFERABLE(i, j, r, m, h)
24:         if th > 0 then
25:             t ← TRANSFER(i, j, th, h, s, ts)
26:             h'_{i+r_1,j+r_2} ← h'_{i+r_1,j+r_2} + t
27:             height_left ← height_left − t
28:         end if
29:     end for
30:     h'_ij ← max(0, height_left)
31: end procedure
```

The pseudo-code CCA is presented in the Algorithm 1. Input parameters to the kernel are i and j, which represent the cell indices, m which represents the model of

the terrain, h which is the height of the fluid, u and v which are the vertical and horizontal velocity, h' the output height, and Θ which represents the minimal allowed height in the cell. It is structured as follows: First it is checked to see if the indices of the thread are inside the bounds of the input array (lines 1–4). Then it is ensured that the cell contains enough mass (at least Θ) and that it is moving with high enough velocity (lines 5–8). The slopes of the neighbouring cells and their total are calculated (lines 9–18). This information is then used to distribute a specific amount of mass (lines 19–29). For each neighbour the maximum amount of mass that can be transferred without it moving uphill is determined, then if this value is positive a portion of this mass, determined by the steepness of the slope, is moved to the neighbouring cell. Mass that was left over is added at the same position in the results array.

Algorithm 2 SWE

1: **procedure** KERNEL4(i, j, m, h, u, v, λ, Δx, Δy, ε, g, a, h', u', v')
2: **if** OUTOFBOUNDS(i, j) **or** $h_{ij} < \varepsilon$ **then**
3: **return**
4: **end if**
5: $F_1^h, F_2^h, G_1^h, G_2^h \leftarrow$ FLUXH(i, j, m, h, u, v, λ)
6: $F_1^{uh}, F_2^{uh}, G_1^{uh}, G_2^{uh} \leftarrow$ FLUXUH(i, j, m, h, u, v, λ)
7: $F_1^{vh}, F_2^{vh}, G_1^{vh}, G_2^{vh} \leftarrow$ FLUXVH(i, j, m, h, u, v, λ)
8: $S_1, S_2 \leftarrow$ SOURCE(i, j, m, h, u, v, Δx, Δy, g, a)
9: $h'_{ij} \leftarrow h_{ij} - \Delta x \cdot (F_2^h - F_1^h) - \Delta y \cdot (G_2^h - G_1^h)$
10: **if** $h'_{ij} \neq 0$ **then**
11: $u'_{ij} \leftarrow u_{ij} - (\Delta x \cdot (F_2^{uh} - F_1^{uh}) + \Delta y \cdot (G_2^{vh} - G_1^{vh}))/h'_{ij} + S_1$
12: $v'_{ij} \leftarrow v_{ij} - (\Delta x \cdot (F_2^{uh} - F_1^{uh}) + \Delta y \cdot (G_2^{uh} - G_1^{uh}))/h'_{ij} + S_2$
13: **end if**
14: **end procedure**

The Algorithm 2, presents the pseudo-code SWE. First six input parameters are the same as the ones for the CCA kernel, then follows λ which represents the characteristic velocities, Δx and Δy represent the cell width and height, ε is the minimum amount of height for the SWE to work, g is the gravity, a is the Manning coefficient, h', u', v' represent the output variables. First it is checked that indices are not out of bounds and that enough fluid is present in the current cell for the SWE solver to run (lines 2–4). Then the fluxes and the source term are calculated (lines 5–8). The resulting height is calculated in line 9, which is then checked to see if it is positive, because it is used as a denominator in lines 11 and 12, where the resulting velocities are calculated.

At the end of the simulation loop the results are copied into the input arrays and the resulting height is transferred back to main memory and then rendered on the screen. After the loop is terminated all allocated memory is freed.

Table 1: Run time of 1000 iterations using different implementations in relation to the lateral size of a square grid in terms of number of cells.

Lateral size	GPU [s]	CPU_8 [s]	CPU_1 [s]
1000	66.2506	129.606	382.163
500	17.1727	33.3051	104.594
333	8.45965	13.9932	46.7858
250	4.72635	7.81743	26.4550
200	3.16506	4.92756	17.1912
166	2.38993	3.54443	11.7735
142	1.82582	2.76242	8.26432
125	1.41043	2.04936	6.35155
111	1.07874	1.60661	4.97288
100	0.97592	1.15555	3.90313

3 Results

To demonstrate the performance of the method two typical hypothetical problems where solved. These problems are challenging because they feature large discontinuances and formation of bores, which cannot be handled by naive discretization techniques.

First is the partial dam break problem, which was studied by Fennema et al [5]. The geometry of the problem consist of a basin, which is divided by a dam in the middle as is shown in Figure 2. The basin is filled with water at both ends. The initial water level in the dammed compartment is twice as high as the tail water. It is assumed that a portion of the dam breaks instantaneously, which means there is a large discontinuity in the initial condition. As the water flows though the breach it forms a large wave which propagates and spreads laterally. The water is exchanged between the compartments, until the water levels eventually equalize [5, 1]. The obtained results are in agreement with the results obtained in [5, 1]. For the sake of clarity, the walls are not included in the figure, the reminder of the dam is represented as empty space between the two compartments. The red colour signifies the amount of liquid missing from the dammed area and the blue colour represents the amount of liquid that has accumulated in the tail water.

The second problem is a circular version of the dam break problem. A column of water is bounded by a circular dam from the surrounding water. Upon dam failure, the dam is instantly removed completely. The body of water spreads radially, until it reflects of the domain boundary [1]. The simulated fluid forms recognisable patterns, which can be seen in Figure 3. The results concede with the results obtained in [1]. The blue colour represents the amount the fluid that is protruding above the surrounding water level, while the red colour represents the amount it depresses below it.

The method was also tested in a more natural test case. It was used to simulate a landslide flowing along a digital terrain model (DTM) derived from high resolution Light Detection and Ranging (LiDAR) data. Initially a conical region of terrain was allotted to the landslide, which

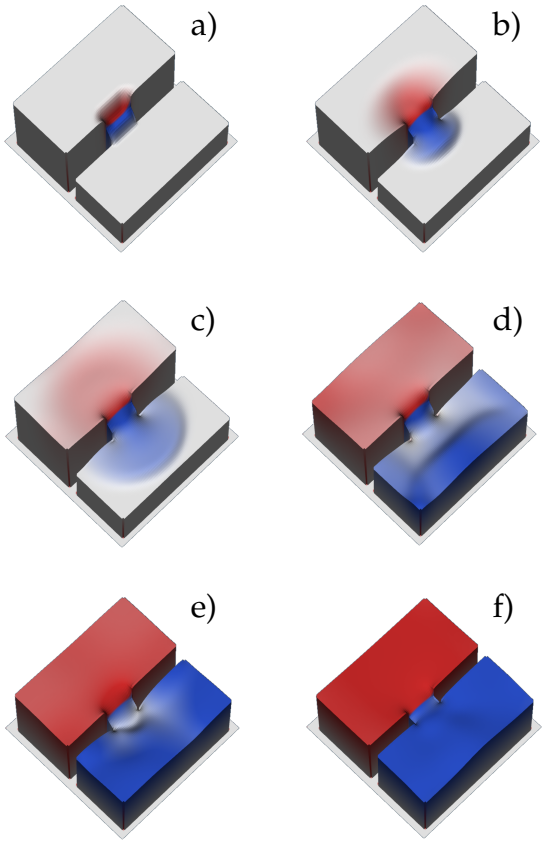

Figure 2: Results of the partial dam break simulation. Times: a) 0.0095, b) 0.0442, c) 0.0838, d) 0.1568, e) 0.2308, f) 0.3086

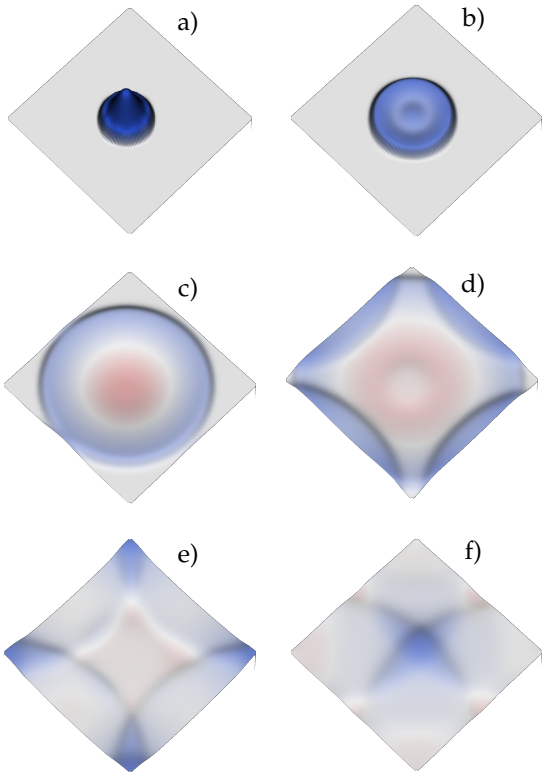

Figure 3: Results of circular dam break simulation. Times: a) 0.0077, b) 0.0227, c) 0.0752, d) 0.1126, e) 0.1501, f) 0.2249.

Table 2: Parameters used in the simulation.

Parameter	Value
ε	1.0
g	9.81
a	0.25
c	0.5
μ	0.1
Θ	0.1
Π	0.1

then moved along the slope due to the effects of the gravity. This thoroughly tested the CCA solution to the wetting/drying problem, since the landslide initially propagates on the dry surface. The results of the simulation are presented in Figure 4. In the Figure 4a the landslide has just moved from the starting point, in Figure 4b it has flowed into the valley, until having reached the edge of the DTM in the Figure 4c. The results are reasonable considering the model is lacking a rheological model, a crucial part of the more advanced landslide models.

The calculated worst-case time complexity of each iteration is $O(n^2)$. It was determined as follows: The worst-case time complexity of the SWE solver is $O(2 \cdot wh)$. In each iteration, the solver passes all cells; given that the w and h represent the width and height of the computation domain, the number of all cells is wh. The algorithm makes two passes, first the field of $|\lambda|$ is calculated, so that the time-step can be evaluated in accordance with the CFL condition. The second pass updates the conservative variables. The time complexity of the CCA is $O(8 \cdot wh)$, all cells are passed and at each position the transfer function is evaluated in eight directions. The total worst-case time complexity of each iteration is, thus, $O(2 \cdot wh + 8 \cdot wh)$. Neglecting the small constant, the estimated time-complexity is $O(wh)$. Considering that the width and the height are usually proportional, the final worst-case complexity is indeed $O(n^2)$. The theoretical derivation is also corroborated by the experimental measurements in Figure 5. As can be observed the general trend is distinctly parabolic for all considered implementations.

The performance of different implementations was evaluated on the dam break test case. Ten different grid sizes where used, created by down-sampling the original mesh of size 1000×1000 grid-points. The parameters that was used in simulation are listed in Table 2. The GPU implementation used enough blocks, each of size 22×22, to cover all cells in the grid. The program was ran on a computer with a Intel Core i7 3610QM processor and NVIDIA GeForce GT 650M graphics card. The results are presented in Table 1 and show clearly that the GPU and multi-threaded implementations offer a large improvement over the single core version, as can be seen in Figure 5. The GPU method performs the best overall, with an average speedup of 1.7 over the multi-threaded version and a 5.5 speedup over the single core version. As can be seen in Figures 6 and 7, the increase in per-

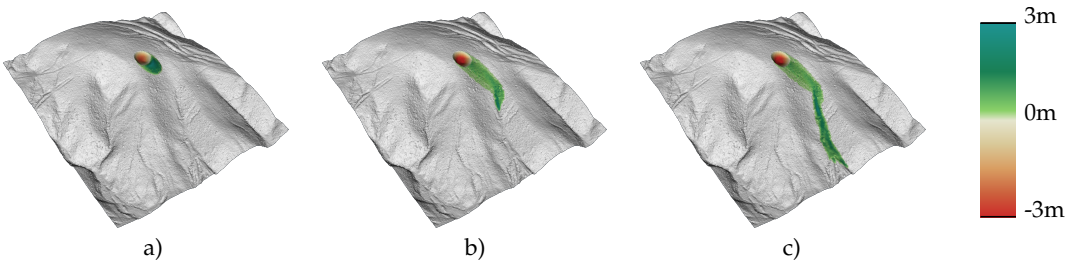

Figure 4: Results of landslide simulation on natural terrain. Iterations: a) 100, b) 1000, c) 3000

formance is raising with the size of the grid, however the variations are minor. Due to the two-dimensional nature of the problem, the time complexity cannot be below $O(n^2)$; therefore, the size of the grid eventually becomes too large to be simulated in real-time. Nonetheless, reasonable mesh sizes have the average iteration time below 70 ms, which is fast enough for human perception to experience it as fluid motion.

The method is relatively stable and the total amount of fluid doesn't fluctuate greatly. Before the simulation of the dam break test case with a 500×500 grid the total height of the fluid equalled 354856 and 354855 after 1000 iterations. The average deviation from the starting height during the simulation was 148.80 and 321.19 the maximum, which is 0.04 % and 0.09 % of the initial total height respectively. The minor fluctuations are probably due to round of errors or other insignificant numerical instabilities.

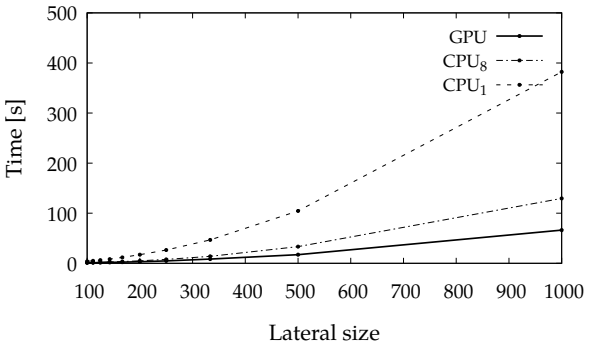

Figure 5: Run time in relation to the lateral size of the square mesh for different implementation.

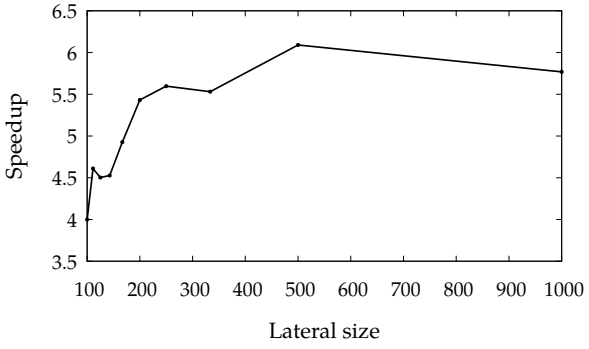

Figure 6: Speedup gained using the GPU instead of the single core CPU implementation.

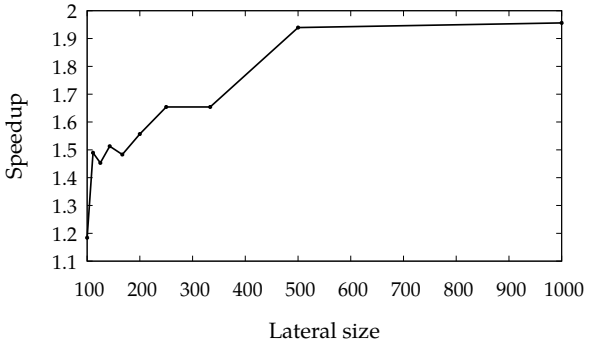

Figure 7: Speedup gained using the GPU instead of the multi-threaded CPU implementation.

4 Conclusion

A novel method for simulating fluid flow has been developed based on a FVM using Lax-Friedrich discretization of SWE and a CCA as a solution of the wetting/drying problem. The method performs in real-time for reasonable mesh sizes and lends itself well to parallelization. It was implemented on the CPU and the GPU using the CUDA platform. The GPU implementation achieved a speedup of 5.5 and 1.7 over the single core and multi-threaded implementation respectively. Lower processing times mean that even larger grids can be simulated in real-time. It was tested successfully on two classical test cases, the partial and circular dam break problem. The results agree with the ones found in the literature. Additionally, the method was evaluated in a more natural test case by simulating a landslide on a DTM. In all tests the method produced plausible and visually appealing results.

References

[1] K. Anastasiou and C. T. Chan. Solution of the 2D shallow water equations using the finite volume method on unstructured triangular meshes. *International Journal for Numerical Methods in Fluids*, 24(11):1225–1245, 1997.

[2] J. D. Anderson and J. Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

[3] S. F. Bradford and B. F. Sanders. Finite-Volume Model for Shallow-Water Flooding of Arbitrary Topography. *Journal of Hydraulic Engineering*, 128(3):289–298, 2002.

[4] L. Dong, L. You-Quan, B. Kai, et al. Real-time shallow water simulation on terrain. In *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry*, pages 331–338. ACM, 2010.

[5] R. J. Fennema and M. H. Chaudhry. Explicit methods for 2-d transient free surface flows. *Journal of Hydraulic Engineering*, 116(8):1013–1034, 1990.

[6] H. P. Gunawan. *Numerical simulation of shallow water equations and related models*. Theses, Université Paris-Est, January 2015.

[7] D. Kleppner and R. Kolenkow. *An introduction to mechanics*. Cambridge University Press, 2013.

[8] R. J. LeVeque. Finite Volume Methods for Hyperbolic Problems. *Cambridge University Press*, 54:258, 2002.

[9] E. Lindholm, J. Nickolls, S. Oberman, et al. Nvidia tesla: a unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, March 2008.

[10] T. H. Pulliam. The Euler Equations. *Notes*:33, 1994.

[11] M. Schäfer. *Computational engineering: introduction to numerical methods*. Springer, 2006.

[12] B. Solenthaler, P. Bucher, N. Chentanez, et al. SPH Based Shallow Water Simulation. In J. Bender, K. Erleben, and E. Galin, editors, *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2011)*. The Eurographics Association, 2011.

[13] J. Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co., 1999.

[14] J. Tan and X. Yang. Physically-based fluid animation: a survey. *Science in China Series F: Information Sciences*, 52(5):723–740, 2009.

[15] H. R. Vosoughifar, H. R. Jalalpour, M. Tabandeh, et al. A high resolution finite volume method for dam break simulation. *SASTech*, m:1–9, 2011.

[16] C. B. Vreugdenhil. Numerical methods for shallow water flow. *VKI, Computational Fluid Dynamics, Volume 1 48 p (SEE N90-27989 22-34)*, 1, 1990.

[17] C. Woolley. Cuda overview. *Developer Technology Group, NVIDIA Corporation. Available online: http://www. cc. gatech. edu/~ vetter/keeneland/tutorial-2011-04-14/02-cuda-overview. pdf (accessed on 10 September 2015)*, 2011.