Article

Not peer-reviewed version

Just-in-Time Fluid Flow Simulation on Mobile Devices Using OpenVisFlow and OpenLB

Dennis Teutscher * , Adrian Kummerländer , Fedor Bukreev , Marcio Dorn , Mathias J. Krause

Posted Date: 31 January 2024

doi: 10.20944/preprints202401.2184.v1

Keywords: CFD; LBM; mobile device; just-in-time; digital twin

Article

Just-in-Time Fluid Flow Simulation on Mobile Devices Using OpenVisFlow and OpenLB

Dennis Teutscher [1,*], Adrian Kummerländer [1] and Fedor Bukreev [1], Marcio Dorn [2] and Mathias J. Krause [1]

[1] Lattice Boltzmann Research Group, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany; dennis.teutscher@kit.edu (D.T.); adrian.kummerlaender@kit.edu (A.K.); fedor.bukreev@kit.edu (F.B.); mathias.krause@kit.edu (M.J.K.);

[2] Structural Bioinformatics and Computational Biology Lab (SBCB), Federal University of Rio Grande do Sul (UFRGS), Brazil; mdorn@inf.ufrgs.br (M.D.)

[*] Correspondence: dennis.teutscher@kit.edu

Abstract: The present state of research in Computational Fluid Dynamics (CFD) is marked by an ongoing process of refining numerical methods and algorithms with the goal of achieving accurate modeling and analysis of fluid flow and heat transfer phenomena. Remarkable progress has been achieved in the domains of turbulence modeling, parallel computing, and mesh generation, resulting in heightened simulation precision when it comes to capturing complex flow behaviors. Nevertheless CFD faces a significant challenge due to the time and expertise needed for meticulous simulation setup and intricate numerical techniques. To surmount this challenge, we introduce *paint2sim*—an innovative mobile application designed to enable on-the-fly 2D fluid simulations using a device's camera. Seamlessly integrated with *OpenLB*, a high-performance Lattice Boltzmann-based library, *paint2sim* offers accurate simulations. The application leverages the capabilities of the Lattice Boltzmann Method (LBM) to model fluid behaviors accurately. Through a symbiotic interaction with the open-source *OpenCV* library, *paint2sim* can scan and extract hand-drawn simulation domains, affording the capability for instant simulation and visualization. Notably, *paint2sim* can also be regarded as a digital twin, facilitating *just-in-time* representation and analysis of 2D fluid systems. The implications of this technology extend significantly to both fluid dynamics education and industrial applications, effectively lowering barriers and rendering fluid simulations more accessible. Encouragingly, the outcomes of simulations conducted with *paint2sim* showcase promising qualitative and quantitative results. Overall, *paint2sim* offers a groundbreaking approach to mobile 2D fluid simulations, providing users with *just-in-time* visualization and accurate results, while simultaneously serving as a digital twin for fluid systems.

Keywords: CFD; LBM; mobile device; just-in-time; digital twin

1. Introduction

Computational fluid dynamics (CFD) has been a vital tool for understanding fluid behavior across various industries and academic domains. However, the practical application of CFD has been limited by the long computational time and complexity involved in building a simulation setup. In recent years, the integration of CFD into various software, including CAD, 3D computational graphic software like *Blender* [1], and game engines like *Unity* [2] and *Unreal Engine* [3], has become more prevalent due to its usefulness in observing fluid flow behavior.

Various researchers have proposed innovative methods for applying CFD to different areas of interest. Mathias Berger and Verina Cristie [4] proposed using game engine technology to bridge the gap between architects and engineers in evaluating the effect of buildings on urban climate through CFD methods. Jos Stam [5] presented a rapid implementation of a fluid dynamics solver for game engines based on the physical equations of fluid flow, emphasizing stability and speed for *just-in-time* performance. Wangda Zuo and Qingyan Chen [6] proposed the Fast Fluid Dynamics (FFD) method as an intermediate approach between nodal models and CFD, providing much richer flow information while being 50 times faster than CFD for conducting faster-than-just-in-time flow simulations for emergency management in buildings. Angela Minichiello et al. [7] introduced a mobile instructional particle image velocimetry (mI-PIV) tool for smartphones and tablets running Android that provides guided instruction to learners, enabling them to visualize and experiment with authentic flow fields in real time. Jia-Rui *at al.* [8] explore the use of augmented reality (AR) technology on mobile devices for visualizing and interacting with CFD simulation results in the context of indoor thermal environment design. Harwood *et al.* [9] developed a GPU-accelerated, interactive simulation framework suitable for mobile devices, enabling visualization of flow around particles.

The Lattice Boltzmann Method (LBM), a versatile computational technique for simulating fluid flow, is central to our approach. LBM has gained popularity due to its ability to handle complex geometries, multiple phases, and mesoscale phenomena. It operates on a lattice grid and models fluid behavior using probability distribution functions, making it suitable for parallel processing on various platforms. LBM has evolved with a variety of lattice structures and collision models, each tailored for specific flow scenarios. Recent advancements include the incorporation of multiple-relaxation-time schemes for improved stability and efficiency, as well as extensions to simulate thermal and multiphase flows.

Previously mentioned work [9] utilizes a static domain with fixed inlets and outlets to create 2D simulations on tablets, which is limited to NVIDIA GPU-based devices.

In this paper, we present a new application called *paint2sim*, which extends the capabilities of *OpenVisFlow* [10], a visualization library that introduced novel solutions to the challenges of long computational time and complexity when targeting mobile devices. Leveraging the power of LBM, *paint2sim* utilizes the open-source library *OpenCV* [11] to enable on-the-fly, *just-in-time* 2D simulations using the camera of a mobile device. Our approach is not limited to specific NVIDIA GPU-based mobile devices but is applicable to all Android devices. Furthermore, we incorporates AR capabilities through the scanning of physical objects. Moreover, in this paper, we compare the performance, demonstrating better results even while utilizing only one core.

Our aim of this approach is to enable users to generate a digital twin of a fluid domain using hand-drawn sketches, effectively converting their mobile devices into virtual laboratories for fluid dynamics. The objective is to facilitate the scanning of a simulation domain and provide real-time visualization of calculated results *just-in-time*, eliminating the need for precompiled simulations or specialized expertise. By seamlessly connecting physical sketches with real-time simulations, *paint2sim* strives to function as a digital twin for 2D fluid flow simulations.

Our contributions include the integration of the Lattice Boltzmann-based library *OpenLB* [12] into the mobile device, *just-in-time* simulation and visualization, as well as stable simulations for most cases. The application *paint2sim* plays a pivotal role in advancing applied CFD by providing students and engineers with a user-friendly platform for quick insights into 2D fluid dynamics. The unique feature of generating *just-in-time* simulations on mobile devices empowers users to swiftly visualize and analyze fluid behavior in real-time, enhancing the accessibility and efficiency of fluid flow studies.

This technology has the potential to revolutionize how we teach and learn about fluid dynamics, as well as how we design and optimize fluid-based systems. *paint2sim* has the potential to benefit a wide range of users and applications, from students learning the fundamentals of fluid dynamics to engineers designing complex systems in the chemical, aerospace, and automotive industries. The technology can also be applied in medical research and environmental studies, where fluid behavior plays a crucial role. By simplifying the simulation process and making it more accessible, *paint2sim* has the potential to democratize the field of CFD and encourage a wider range of users to explore the fascinating world of fluid dynamics.

In the remainder of this paper, we delve into the method employed by *paint2sim*, present the numerical results obtained through its implementation, and engage in a comprehensive discussion of these results.

2. Method

There are three critical requirements that must be fulfilled in order to run scanned hand-drawn simulation domains and simulate them locally on mobile devices: high performance to simulate and visualize the simulation *just-in-time*, high stability due to the various domains that can be scanned and the adjustable Reynolds number. At last the physical accuracy should have quantitatively minimal errors and should be qualitatively comparable to reality.

In the following sections, we discuss the LBM, the simulation model used in this study, and its suitability for high performance and physical accuracy. Following that, we describe the methods utilized to ensure the stability of the simulation.

2.1. Lattice Boltzmann Methods (LBM)

LBM offers several advantages when it comes to both performance and physical accuracy in simulating fluid behavior. One advantage is that LBM can be easily parallelized, allowing for faster computation times and higher performance. Another advantage is that LBM inherently models the fluid at a mesoscopic level, allowing for accurate representation of complex physical phenomena such as turbulence and multiphase flows. This makes LBM well-suited for simulating a wide range of fluid dynamics problems. In the remainder of this section, we provide a brief introduction to LBM and the target equations, the Navier-Stokes equations (NSE) for mass and momentum conservation in fluid dynamics.

The NSE is the fundamental equation that governs the behavior of fluids, and it is widely used in CFD simulations. The NSEs in full form can be solved only numerically using various discretization methods, such as the finite difference, finite volume method or LBM.

The NSEs can be written as:

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{\nabla p}{\rho} + \nu_m \nabla^2 \mathbf{u} + \frac{\mathbf{F}}{\rho}, \tag{2}$$

where $\mathbf{u}$ is the velocity vector, p is the pressure, ρ is the fluid density, ν_m is the molecular kinematic viscosity of the fluid, and $\mathbf{F}$ is the external force acting on the fluid.

The LBM approximates the conservation equations in its limit (Chapman-Enskog expansion) on a discrete grid of points connected by a set of links that represent the paths along which the fluid particles can move [13]. In LBM, the spatial and temporal state of these particles is represented by the probability distribution functions (PDFs) that evolve according to the lattice Boltzmann equation

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = \Omega^C + \Omega^F, \tag{3}$$

where f_i is the PDF at lattice node i and time t, $\mathbf{e}_i$ is the normalized discrete velocity in the i-th direction, $\Omega^C = -(f_i - f_i^{eq})/\tau$ is the collision operator and Ω^F is the Guo forcing term [14].

In the current simulations, the D2Q9 lattice is used, where D is number of dimensions and Q is number of the normalized discrete velocity directions. The corresponding lattice cell is shown in Figure 1.

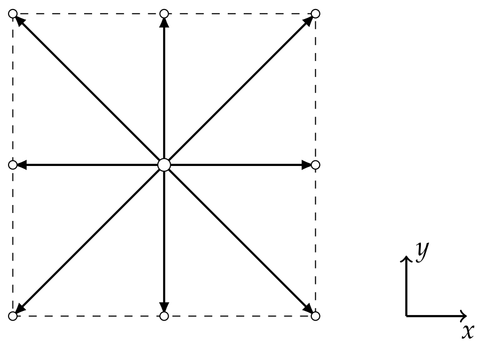


Figure 1. A schematic illustration of the discrete velocity set for the $D2Q9$ lattice

2.2. Smagorinsky BGK Collision Model

The Smagorinsky model [15] is a subgrid-scale model used in large eddy simulation (LES) of turbulent flows. The model introduces a turbulent viscosity term to the governing equations of fluid flow, which is based on the strain rate tensor of the flow field. The model filters out the small unresolved vortices by replacing them with an artificial viscosity increase. The large eddies are preserved. The modified strain rate tensor describes the production of turbulent kinetic energy in the flow.

The turbulent viscosity term is given by:

$$\nu_t = (C_S \triangle x)^2 |\mathbf{S}|, \tag{4}$$

where ν_t is the turbulent eddy viscosity, $\triangle x$ is the grid spacing, C_S is the Smagorinsky constant, and $|\mathbf{S}|$ is the magnitude of the strain rate tensor. The Smagorinsky constant is the filtering parameter that determines which eddies are neglected.

The modified momentum equation with the addition of the turbulent viscosity term becomes:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu_{eff} \nabla^2 \mathbf{u} + \frac{\mathbf{F}}{\rho}, \tag{5}$$

$$\nu_{eff} = \nu_m + \nu_t. \tag{6}$$

For incompressible NSEs with Smagorinsky LES approach, the lattice Boltzmann equation using the BGK collision operator [16] can be rewritten as:

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i \Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = -\frac{\Delta t}{\tau_{eff}(x,t)} (f_i - f_i^{eq}) + \Omega^F, \tag{7}$$

where $\tau_{eff}(x,t) = \frac{\nu_{eff}(x,t)}{c_s^2} \frac{\triangle t}{\triangle x^2} + \frac{1}{2}$ is the effective relaxation time adapted to Smagorinsky model. Here, c_s is the discrete speed of sound.

Due to obstacles in the path of fluid, flow instabilities can be induced even at low Reynolds numbers. Therefore, it is necessary to adjust the relaxation time accordingly. The Smagorinsky BGK model accomplishes this by automatically increasing the relaxation time at the cells with a high shear rate. In the case of a laminar flow, the turbulent viscosity $\nu_t \approx 0$. Choosing a correct Smagorinsky constant secures stable run of the simulation.

2.3. Fringe Region Technique

A fringe region technique [17] is used to eliminate instabilities at the outflow boundary condition. The outlet can become divergent, if a large eddy flows through it. In order to compensate this, a fringe zone is applied to laminarize the outflow. To achieve this, the NSE in the near-to-outlet region is forced with a special term

$$\mathbf{F} = \lambda(x) \cdot (\mathbf{U} - \mathbf{u}), \tag{8}$$

where $\mathbf{U}$ is the prescribed velocity, $\lambda(x)$ is the fringe function that varies smoothly from 0 to 1 over a distance of a few grid points, and $\mathbf{u}$ is the computed velocity.

In the fringe region technique, the prescribed velocity $\mathbf{U}$ is obtained using a mixing length model. The mixing length model can be written as:

$$\mathbf{U}(x) = \mathbf{U}(x) + [\mathbf{U}(x_{out}) - \mathbf{U}(x)] S\left(\frac{x - x_{mix}}{\Delta_{\text{mix}}}\right), \tag{9}$$

where $\mathbf{U}(x)$ is the velocity at a point x, x_{out} is the outlet coordinate, x_{mix} and Δ_{mix} are tuning parameters for transition between real and prescribed velocity, and S is a smooth function that varies from 0 to 1 over a distance.

2.4. Concept and Realization

The implementation of the system involves two separate shared libraries: one for *OpenCV* and one for *OpenLB*. *OpenCV* provides the image processing capabilities, while *OpenLB* provides the CFD simulation capabilities. The shared libraries are written in C++ and can be compiled on a variety of platforms. The implementation involves creating a set of functions that can be used by both *OpenCV* and *OpenLB* libraries. These functions are used to perform image processing and CFD simulation, respectively. In addition, *Unity* is used to create the application for the mobile device. Both *OpenCV* and *OpenLB* communicate independently with *Unity* through their respective shared libraries.

2.4.1. Concept

In order to achieve Just-In- Time simulation and visualization, there must be a clear seperation between the frontend which is the application itself and the backend which consist of the *OpenLB* and *OpenCV* shared libraries. Each of the shared libraries are called in seperate threads which allows for decoupling of simulation and visualization thereof. The communication between the frontend and backend consists primarily of the exchange of the simulation results for a timestep in the form of a pressure or velocity array as shown in Figure 2.

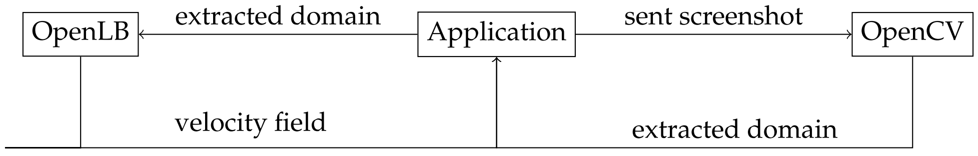


Figure 2. Communication between the Application, *OpenCV* and *OpenLB*

2.4.2. Structure of the *OpenLB* Shared Library

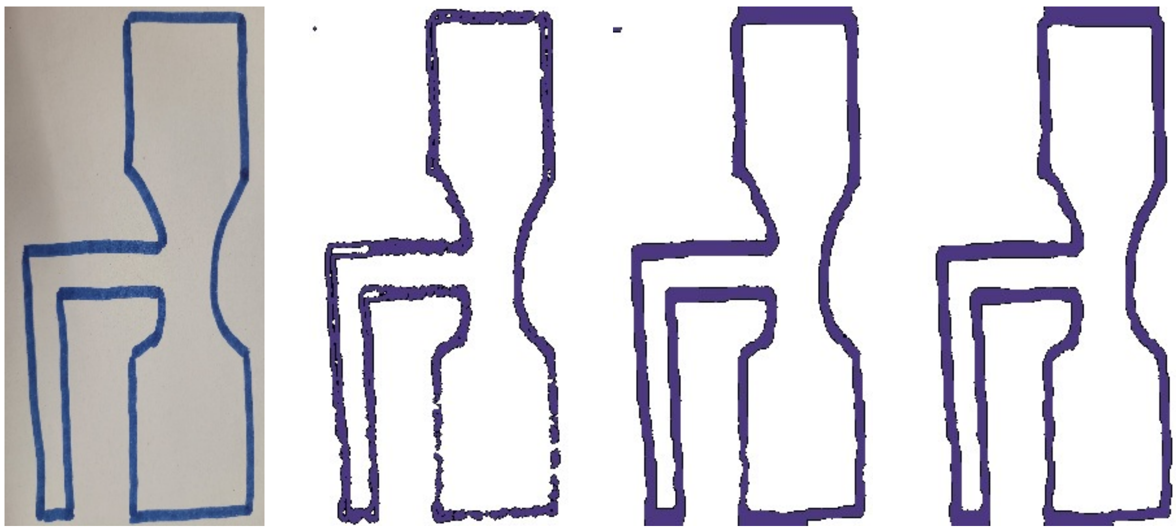
In this section, we present an overview of the *OpenLB* shared library structure that we employ for simulating on smartphones. Specifically, Algorithm 1 illustrates the main loop of the library, which adheres to the standard Lattice Boltzmann simulation structure with *OpenLB*. Initially, *OpenLB* is instantiated, and crucial classes are initialized. The unit converter, which stores the lattice relevant data, is then declared. Next, the simulation domain is instantiated, and its dimensions correspond to the domain scanned with the smartphone. This domain is then passed to the load balancer, which distributes the cuboid into subsections necessary for parallel computing. The material number map is an array of numbers that refer to the materials in the simulation. However, it cannot be utilized as is and necessitates transfer to the *OpenLB*-specific class, `superGeometry`. After preparing the geometry, the lattice is ready for simulation, and boundary conditions can be set. In particular, we apply the Smagorinksy-BGK model Section 2.2 to the material numbers of the fluid, outflow, and inflow. Additionally, we utilize the fringe region technique around the outflow. Moreover, we need to specify which material numbers define the inlet and outlet. We also add the postprocessor, which receives relevant dimensions and pointers to the result arrays. At step 10 of the Algorithm 1, the simulation commences with the **for**-loop. In this loop, T corresponds to the total simulation time, while iT represents the current timestep. For each timestep, we update and set the boundary values. Subsequently, we call the collide and stream functions to retrieve the values for the subsequent step. Finally, we synchronize the number of timesteps saved per second with the frames per second (t_{fps}) of the smartphone application (step 13 to 21). This synchronization ensures better performance as we do not save every timestep of the simulation, but still maintain a fluid visualization for the user

Algorithm 1 Mainloop of the *OpenLB* Shared Library

```
1: init OpenLB
2: declare unit converter
3: instantiation of the simulation domain
4: instantiation of a load balancer
5: preparing of the geometry
6: preparing of the lattice
7: add postprocessor
8: calculate the number of timesteps from the total simulation time $T$
9: start timer $t_{fps}$
10: for $iT = 0; iT \leq T; i++$ do
11:     set boundary values
12:     collide and stream
13:     if $t_{fps} \leq 1$ then
14:         if $\Delta t \leq 1/fps$ & $count \leq fps$ then
15:             write results via postprocessor
16:             write Mega Lattice Updates per Second
17:         end if
18:         count++
19:     end if
20:     reset $t_{fps}$
21:     count = 0;
22:     if endSimulation then break;
23:     end if
24: end for
```

2.4.3. Structure of the *OpenCV* Shared Library

The *OpenCV* shared library plays a critical role in enabling *OpenVisFlow* to extract contours, which is a vital step in obtaining the simulation domain from an image. Contour extraction involves identifying the object's boundary in an image and approximating it with a curve. The curve consists of continuous points along the boundary with the same color or intensity. The process of domain extraction begins by resizing the image to the desired simulation resolution, followed by applying a threshold to enhance the contrast between the domain and the background. The `findContours` function is then utilized to extract the domain boundaries. Finally, morphological functions are applied to post-process the extracted domain. Specifically, a morphological close function is used to fill potential holes in the boundary, while a morphological open function is used to eliminate noise from the domain. Figure 3 displays the results of the processing steps, where Figure 3(a) depicts the photo of the hand-drawn domain to be extracted, Figure 3(b) presents the outcome of the initial extraction with a threshold, Figure 3(c) and Figure 3(d) represent the post-image processing steps.



(a) Photo of the drawn domain. **(b)** Extracted contour **(c)** morphological close **(d)** morphological open

Figure 3. The figure illustrates the domain extraction process, starting with the original image (a), followed by the resulting image after the initial extraction with a threshold (b). Post-image processing steps are then applied, leading to the final processed images shown in (c) and (d).)

2.5. Expanding OpenVisFlow for Mobile Fluid Flow Simulation with OpenLB

The *OpenVisFlow* library, based on *Unity*, is designed to be easily expandable, allowing it to handle and visualize various data types. To achieve this, two new classes are required that inherit from the parent `DataManager` class. These classes will enable communication between *OpenVisFlow* and the shared libraries *OpenLB* and *OpenCV*. Figure 4 depicts the class diagram for the new classes, `OpenCVDatamanger` and `OpenLBDatamanger`, which inherit from the parent class `Datamanager`.

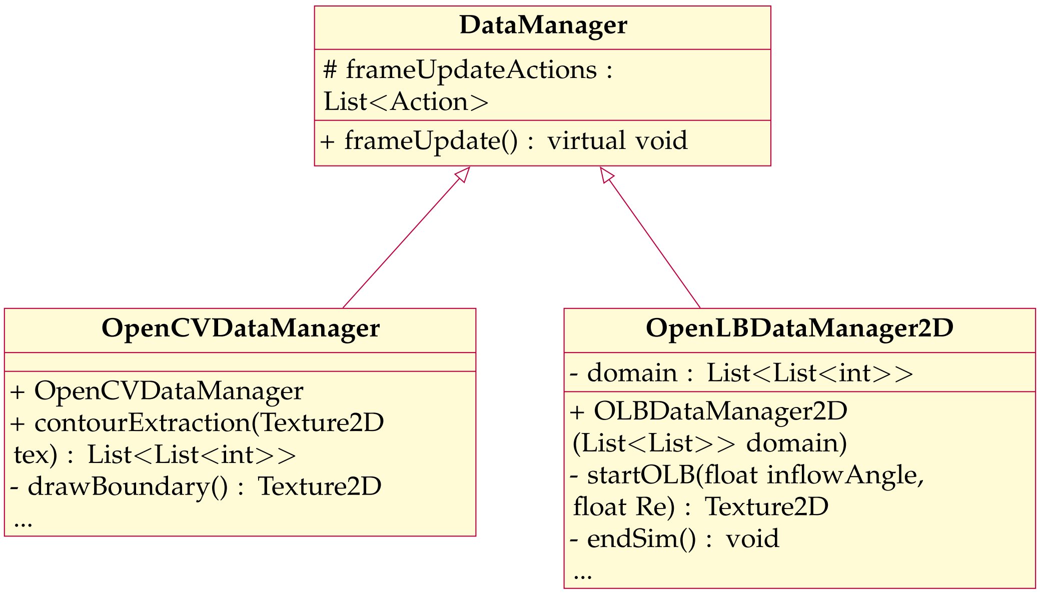
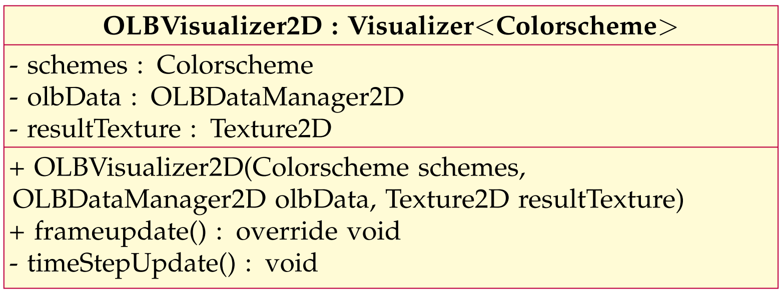

DataManager
# frameUpdateActions : List<Action>
+ frameUpdate() : virtual void
OpenCVDataManager
+ OpenCVDataManager
+ contourExtraction(Texture2D tex) : List<List<int>>
- drawBoundary() : Texture2D
...
OpenLBDataManager2D
- domain : List<List<int>>
+ OLBDataManager2D (List<List>> domain)
- startOLB(float inflowAngle, float Re) : Texture2D
- endSim() : void
...

Figure 4. Class diagram for the *OpenCV* and *OpenLB* DataManager, highlighting the most essential functions.

The `OpenCVDataManager` class adds necessary functions for communication between *OpenVisFlow* and the *OpenCV* shared library. Similarly, the `OpenLBDataManager2D` class inherits from `DataManager` and includes the essential functions required to initiate and terminate a simulation. The class also contains additional functionalities such as optional placement of a fringe zone. Overall, these new classes ensure seamless communication between *OpenVisFlow* and the *OpenLB* and *OpenCV* shared libraries, allowing for efficient data handling and visualization.

2.5.1. Visualization of the Simulation Data

In order to visualize the calculated results of the in section 2.4.2 introduced *OpenLB* shared library. A new visualizer class has to be implemented which has the ability to handle 2D arrays of the type float, transform this information to a texture and display it. The class diagram of said new class is depicted in Figure 5. Following the *OpenVisFlow* framework, the visualizer is initialized with a `Colorscheme`, which consist of different colors that are used to visualize the flow and an instance of the `OLBDataManager2D` in order to get the simulation data. The function `timeStepUpdate` is added to the inherited action list. This function is called on every frame and extracts the minimum and maximum bounds for the macroscopic moments of the current timestep. Based on that, the color scheme can be mapped to each cell and rendered to a texture for display.



OLBVisualizer2D : Visualizer<Colorscheme>

- schemes : Colorscheme
- olbData : OLBDataManager2D
- resultTexture : Texture2D

+ OLBVisualizer2D(Colorscheme schemes, OLBDataManager2D olbData, Texture2D resultTexture)
+ frameupdate() : override void
- timeStepUpdate() : void

Figure 5. Class diagram of the *OLBVisualizer2D*, highlighting the most essential functions.

2.6. User Guide for paint2sim

The application *paint2sim* is available for download at www.openLB/paint2sim/. This section provides a detailed description of how to use the application. The forward/backward buttons are used to navigate between different steps, while the cross can be used to reset the application. All steps are shown in the remaining part of this section. It is advisable to use a thick fine-tip marker to draw the domain for effective domain recognition.

Selecting Resolution: Begin by selecting your preferred resolution using the slider, as illustrated in Figure 6(a). Options range from 100 to 300 in 50-unit increments, representing the number of cells across the screen width on your mobile device. Note that higher resolutions may impact user experience on certain devices, potentially causing performance issues.

Scanning the Domain: Utilize the scan button (Figure 6(b)) to initiate the domain scanning process. Once completed, input the scale of the domain in meters (Figure 6(c)).

Setting Characteristic Length: Set the characteristic length for the Reynolds number estimation either through the input field or by adjusting the scale via touch (Figure 6(d)).

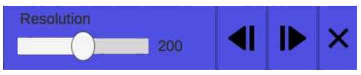
$$Re = \frac{u \cdot l_c}{\nu}$$

Edit the Domain: The domain can be edited using various touch-based options as shown in Figure 6(e). Edit function are: Add walls; Remove existing walls; Declare particles for drag and lift calculations (currently limited to one particle); Define mandatory inlets and outlets

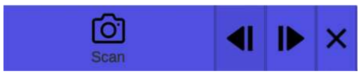
Fluid Parameters: Choose fluid parameters from the dropdown menu, selecting Water, Oil, or Air. Alternatively, input custom kinematic viscosity and density values (Figure 6(f)).

Reynolds Number: Enter the Reynolds number for the simulation. (Figure 6(h)).

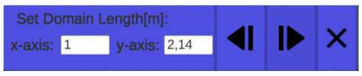
Inflow Direction: If not opting for a Poiseuille inflow, enter the desired inflow direction (Figure 6(g)).


Resolution 200

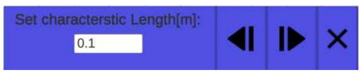
(a) Selecting Resolution



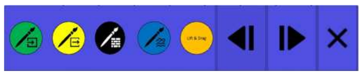
(b) Scanning Domain


Set Domain Length[m]:
x-axis: 1 y-axis: 2,14

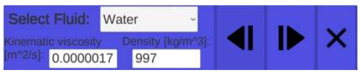
(c) Input Domain scale


Set characterstic Length[m]: 0.1

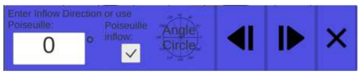
(d) Characteristic length



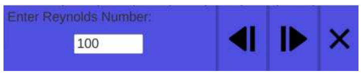
(e) Edit domain


Select Fluid: Water
Kinematic viscosity [m^2/s]: 0.0000017 Density [kg/m^3]: 997

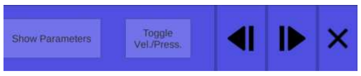
(f) Select fluid


Enter Inflow Direction or use Poiseuille: 0 °
Poiseuille inflow: ✓

(g) Choose inflow


Enter Reynolds Number: 100

(h) Reynolds Number

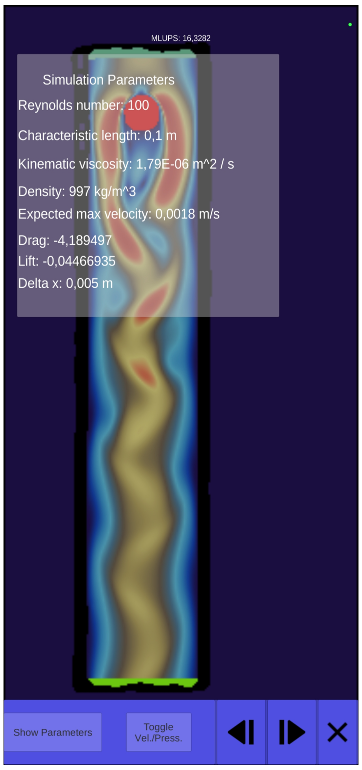


(i) Toggle vizualization

Figure 6. The figures show the input menus of *paint2sim* to scan and set up a simulation

Simulation: The final step initiates the simulation, which runs continuously until manually canceled using the back button or the cross button. During the simulation, toggle between velocity and pressure visualization methods, and view simulation parameters as needed.

If every step was done correctly, the simulation should be running *just-in-time* on the mobile screen. In Figure 7, an example simulation is shown.


MLUPS: 16,3282

Simulation Parameters
Reynolds number: 100
Characteristic length: 0,1 m
Kinematic viscosity: 1,79E-06 m^2 / s
Density: 997 kg/m^3
Expected max velocity: 0,0018 m/s
Drag: -4,189497
Lift: -0,04466935
Delta x: 0,005 m

Figure 7. Resulting example simulation after following each step (Figure 6(a) - 6(i)).

3. Numerical Experiments and Discussion of Results

This section evaluates the precision and performance of LBM as implemented in *paint2sim*. We categorize the results into qualitative and quantitative aspects. The qualitative section focuses on the visualization of the simulation results and their correspondence to physical reality. For the quantitative part, we assess the performance and physical accuracy of the simulation.

3.1. Test Case Setup

To validate the capabilities of *paint2sim*, we chose the 2D cylinder test case provided by the *OpenLB* library. This test case replicates the configuration detailed by Schäfer *et al.* [18]. This selection enables a direct comparison between the results generated by *paint2sim* and the established outcomes in the relevant domain. In addition, we recreated the same scenario through manual drawing, ensuring consistent proportions as outlined in the reference work [18]. Subsequently, the hand-drawn representation was scanned using *paint2sim*, enabling a precise evaluation of its accuracy in replicating the expected flow patterns and attributes. The geometry used for validation is visually presented in Figure 8.

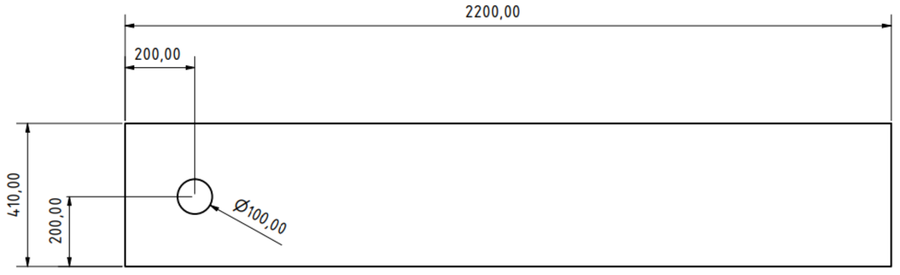


Figure 8. Geometry employed for validation, with dimensions in SI millimeters.

It is important to note that, in this study, *paint2sim* incorporates the Smagorinsky BGK Model and the Fringe Region Technique to maintain simulation stability even at lower resolutions. Notably, the validated cylinder2D case does not utilize either of these techniques

3.2. Choice of Discretisation Parameters

To accommodate various performance capabilities on mobile devices, *pai-nt2Sim* offers four resolution options in terms of Δx, representing the voxel length. In alignment with this, we have conducted corresponding simulations using the *OpenLB* framework. Since we aim to maintain constant time steps in order to be able to ensure visually smooth simulations, the lattice velocity remains consistent within the *paint2sim* application. However, in the case of the *OpenLB* example cylinder2D, the lattice velocity adjusts in accordance with the chosen resolution. To demonstrate that *OpenLB* produces consistent results with those presented in Schäfer *et al.* [18], we also incorporated a higher resolution. Due to the performance restriction inherent to mobile devices, this higher resolution cannot be executed on mobile devices. *paint2sim* has three differences compared to the validation case. The use of the Smagorinsky BGK Collison Modell, the fringe region technique which are used to ensure stability. The application also does not use the second order boundary condition Bouzidi but the first order Bounceback. Reason for this, is that Bouzidi nessesitates the real geometry surface, which is difficult to extract from an already discretized scanned domain. In order to see the influence of the applied techniques, we prepared the following test cases with *paint2sim* which are explained in the remaining part of this section.

paint2sim-1 The fringe region technique as well as the Smagorinsky BGK Collision Model are used with Smagorisnky Constant $C_s = 0.15$.

paint2sim-2: The fringe region technique is used while the Smagorinsky BGK Modell is replaced with the BGK Collison Modell.

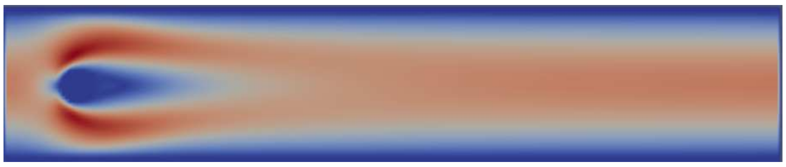
paint2sim-3: The fringe region technique is removed and the Smagorinsky BGK Modell is replaced with the BGK Collison Modell.

In order to have a comparison between Bouzidi and Bounceback. We also ran the validation case from *OpenLB* with Bouzidi ***OpenLB-1*** and with Bounceback ***OpenLB-2***.

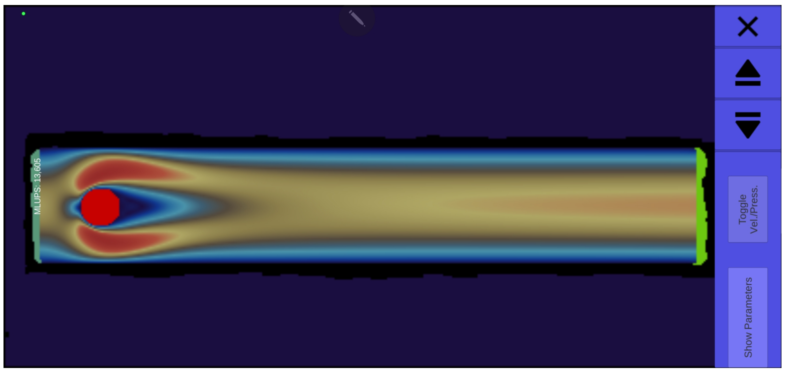
3.3. Validation

3.3.1. Qualitative Results

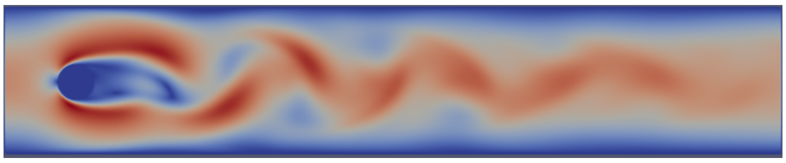
Figure 9 presents a side-by-side comparison of the results. Observing the laminar flow at $Re = 20$, there is no noticeable difference between the validated case shown in Figure 9(a) and the result obtained using *paint2sim* shown in Figure 9(b). Furthermore, when comparing the turbulent flows in Figure 9(c) and Figure 9(d), the qualitative results are also in agreement.



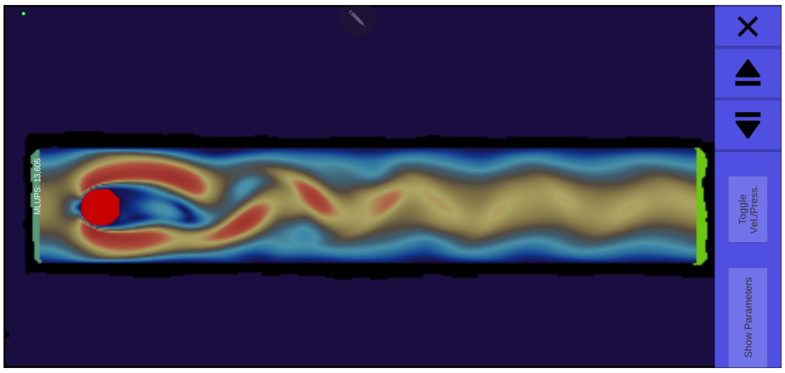
(a) Example of the 2D cylinder from *OpenLB* with $Re = 20$, validated according to Schäfer *et al.* [18].



(b) Simulation using *paint2sim* of a hand-drawn domain with $Re = 20$, replicating the example used for validation.



(c) Example of the 2D cylinder from *OpenLB* with $Re = 100$, validated according to Schäfer *et al.* [18].



(d) Simulation using *paint2sim* of a hand-drawn domain with $Re = 100$, replicating the example used for validation.

Figure 9. Comparison of qualitative results between *OpenLB* and *paint2sim* for cylinder simulations.

3.3.2. Quantitative Results

This section undertakes a comparative assessment of the validation results obtained from [18], as depicted in Table 1, concerning drag and lift coefficients on the cylinder, with both *OpenLB* and *paint2sim*. Table 2 presents the results of simulations conducted at $Re = 20$, detailing the parameters and resulting drag and lift coefficients for specific cases. The results from OpenLB align within the predefined bounds set in Table 1 when utilizing a sufficiently high resolution. Conversely, *paint2sim* at its maximum resolution yields results that deviate by approximately 11% for the drag coefficient. Additionally, the lift coefficient exhibits considerable variation, failing to closely match the specified margin. This discrepancy primarily stems from errors introduced during the hand-drawn domain scan, where image processing techniques for domain extraction introduce slight shape differences, particularly in the representation of the cylinder within the domain. Given the current computational limitations of the mobile devices used for the scan and corresponding simulation resolution, addressing this issue comprehensively is presently impractical. Nevertheless, the results emphasize that an increase in resolution contributes to a reduction in the margin of error.

In Table 3, the outcomes for flow simulations at $Re = 100$ exhibit similar challenges to those encountered in the stable scenario at $Re = 20$.

Table 1. Results of drag and lift coefficients from Schäfer *et al.* [18] in a laminar flow around a cylinder with Re =20 for the stable case and Re= 100 for the unstable case.

Reynolds Number (Re)	20	100
Characteristic Length [m]	0.100	0.100
Voxel Length [m]	-	-
Drag Coefficient	5.570 - 5.590	3.220 - 3.240
Lift Coefficient	0.010 - 0.011	0.990 - 1.010

Table 2. Comparative Analysis of Drag and Lift Coefficients: *OpenLB* vs. *paint2sim* in the Flow Around a Cylinder at $Re = 20$ (Stable Case).

$\Delta x[m]$	OpenLB-1		OpenLB-2		paint2sim-1		paint2sim-2		paint2sim-3	
	Drag	Lift	Drag	Lift	Drag	Lift	Drag	Lift	Drag	Lift
0.010	5.820	0.015	6.116	0.016	6.854	0.038	6.568	0.097	6.625	0.101
0.006	5.689	0.008	5.785	0.008	6.413	0.093	6.354	0.149	6.308	0.157
0.005	5.632	0.012	5.796	0.012	6.319	0.092	6.216	0.030	6.129	0.102
0.004	5.624	0.009	5.740	0.009	6.304	0.002	6.219	0.026	6.232	0.046
0.003	5.601	0.010	5.505	0.010	-	-	-	-	-	-
0.002	5.593	0.010	5.628	0.024	-	-	-	-	-	-

Table 3. Comparative Analysis of Drag and Lift Coefficients: *OpenLB* vs. *paint2sim* in the Flow Around a Cylinder at $Re = 100$ (Unstable Case).

$\Delta x[m]$	OpenLB-1		OpenLB-2		paint2sim-1		paint2sim-2		paint2sim-3	
	Drag	Lift	Drag	Lift	Drag	Lift	Drag	Lift	Drag	Lift
0.010	-	-	-	-	4.892	1.463	4.617	1.518	5.003	1.337
0.006	3.696	1.326	3.988	1.654	4.405	1.213	4.509	1.499	4.662	1.153
0.005	3.485	1.157	3.777	1.449	3.840	0.831	3.721	0.746	3.938	1.051
0.004	3.353	1.089	3.556	1.261	3.344	0.928	3.372	0.903	3.054	1.006
0.003	3.324	1.091	3.361	1.110	-	-	-	-	-	-
0.002	3.264	0.991	3.273	1.013	-	-	-	-	-	-

3.4. Performance

Figure 10 presents the total performance achieved by *paint2sim* measured in millions of cell updates per second (*Mega Lattice Updates per Second (MLUPS)*) across a set of mobile- and stationary test devices.

A central performance bottleneck for numerical simulations on mobile devices due to their inherent compute-heavy nature is given by heat management constraints. This is the primary explanation for visible performance fluctuations as mobile devices tend to aggressively reduce their power output beyond short performance bursts in order to prevent overheating.

While the underlying LBM library *OpenLB* supports various parallelization modes both on CPU and GPU targets [12,19], *paint2sim* explicitly only uses single-threaded single-precision non-vectorized execution for maximum device portability and as a heat-management trade-of. While OpenMP-based shared memory parallelization is possible, core heterogeneity and thread binding caused issues across the diverse set of test devices. Single threaded execution provides sufficient cross-device performance for the intended two-dimensional flow simulations.

On the lowest end, we conducted a performance comparison between *paint2sim* and the results presented in [9]. Due hardware availability issues, we were unable to use the same experimental setup and instead relied on a low-end Huawei P8 Lite with inferior specifications compared to the initially high-end NVIDIA Shield K1 Tablet. Table 4 presents a comparison of the specifications obtained via Geekbench, a well-established mobile benchmark suite, and the achieved total performance in MLUPS. Despite the decision to not utilize parallelization, *paint2sim* performance compares favourably at a speedup of approximately 1.45.

Among the tested mobile devices in Figure 10, the Samsung Galaxy A80 offers the lowest CPU performance. Despite this, the visualization remains smooth for users at an average throughput of 10 MLUPS and no noticeable lags. At the top end, the single-threaded LBM performance on a Samsung Galaxy S22 Ultra is quite close to unvectorized performance on higher-end x86 CPU cores at approximately 24 MLUPs.

Overall, the performance characteristics exhibited by *paint2sim* are sufficient for the intended simulation cases and highly competitive to single cores of full-powered x86 CPUs considering the comparably smaller power envelope.

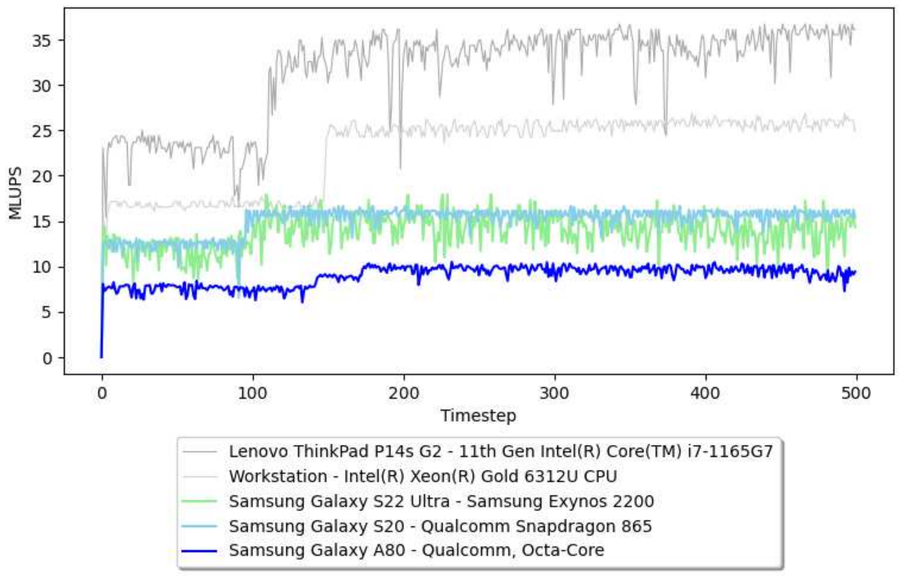


Figure 10. Performance Comparison of *paint2sim* Across Different Devices. Performance is measured in Mega Lattice Updates per Second (MLUPS).

A notable issue associated with mobile devices is their tendency to decrease the power output of the CPU in order to prevent overheating. Figure 11 presents the average MLPUs of the mobile device *Samsung Galaxy S22 Ultra* during extended simulation periods. The graph illustrates a significant decline in performance over time. Moreover, the decline is not continuous; rather, the device maintains its performance until the temperature reaches a critical point, at which it then ramps down to a lower performance level.

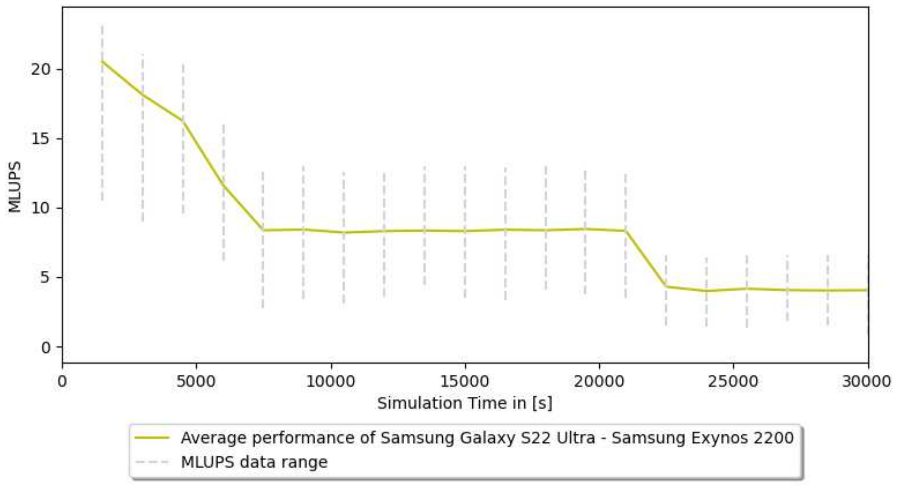


Figure 11. Performance decline over time of *paint2sim*. Performance is measured in Mega Lattice Updates per Second (MLUPS).

Table 4. Performance and spec comparison of Nvidia Shield Tablet K1 and Huawei P8 Lite. For the spec comparison and Single-Core Score Geekbench [20]

Aspect	Nvidia Shield Tablet K1	Huawei P8 Lite
Processor	ARM tn8	ARM ARMv8
Base Frequenzy	**2.22** GHz	1.71 GHz
Single-Core Score	**207**	166
MLUPS	1.1	**1.6**

3.5. Conclusions

In this paper, we presented the *paint2sim* software for LBM simulations on the mobile devices and the numerical validation of the software on example of the Schäfer test case. The simulation results were categorized into qualitative and quantitative aspects, focusing on visualization, performance, and physical accuracy. Regarding performance, we analyzed the total performance achieved by *paint2sim* on various mobile devices. The Samsung Galaxy A80 exhibited the lowest CPU performance among the tested devices but still provided smooth visualization with an average of 10 Mega Lattice Updates per Second (MLUPS) and no noticeable lags. On the other hand, the Samsung Galaxy S22 Ultra demonstrated impressive performance, comparable to higher-end x86 CPUs, achieving the highest MLUPS of 24 under specific conditions. However, fluctuations in performance were observed due to heat management constraints, leading to aggressive reduction of CPU power output beyond short performance bursts.

The validation of *paint2sim* involved qualitative and quantitative comparisons. For qualitative validation, we replicated a well-established 2D cylinder example using a hand-drawn image and compared the results with the validated case from *OpenLB*. The flow patterns and characteristics exhibited by *paint2sim* were found to be in agreement with the established findings. For quantitative validation, we compared the drag and lift coefficients of the cylinder simulation with the results from previous studies. While the drag coefficients showed minor discrepancies of around 10%, the lift coefficients obtained from *paint2sim* differed greatly. These variations can be attributed to the shape approximation of the hand-drawn cylinder and slight inaccuracies in the scanned image.

Overall, the performance and accuracy of *paint2sim* on mobile devices proved to be sufficient for the intended simulation cases and highly competitive with full-powered x86 CPUs within a smaller power envelope. Further enhancements can be achieved through parallelization of the application on mobile devices, which would significantly improve performance metrics. Future work should also address the issue of performance fluctuations resulting from heat management constraints. Taking these factors into account, the digital twin and virtual laboratory features of *paint2sim* hold the promise of offering a valuable simulation tool for 2D computational fluid dynamics applications, allowing for on-the-go simulations.

Author Contributions: Conceptualization, D.T.; methodology, D.T.; software, D.T.; validation, D.T.; formal analysis, D.T.; investigation, D.T.; resources, M.J.K.; data curation, D.T.; writing—original draft preparation, D.T.; writing—review and editing, A.K., F.B., M.D and M.J.K.; supervision, M.J.K.; project administration, M.J.K.; funding acquisition, M.J.K.. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by The Ministry of Science, Research and the Arts of the State of Baden-Württemberg, grant number 34-7811.553-4/5, and by the Lattice Boltzmann Research Group

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Community, B.O. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.
2. Haas, J.K. A history of the unity game engine **2014**.
3. Epic Games. Unreal Engine.
4. Berger, M.; Cristie, V. CFD post-processing in Unity3D. *Procedia Computer Science* **2015**, *51*, 2913–2922.
5. Stam, J. Real-time fluid dynamics for games. In Proceedings of the Proceedings of the game developer conference, 2003, Vol. 18, p. 25.
6. Zuo, W.; Chen, Q. Real-time or faster-than-real-time simulation of airflow in buildings. *Indoor air* **2009**, *19*, 33.
7. Minichiello, A.; Armijo, D.; Mukherjee, S.; Caldwell, L.; Kulyukin, V.; Truscott, T.; Elliott, J.; Bhouraskar, A. Developing a mobile application-based particle image velocimetry tool for enhanced teaching and learning in fluid mechanics: A design-based research approach. *Computer applications in engineering education* **2021**, *29*, 517–537.
8. Lin, J.R.; Cao, J.; Zhang, J.P.; van Treeck, C.; Frisch, J. Visualization of indoor thermal environment on mobile devices based on augmented reality and computational fluid dynamics. *Automation in Construction* **2019**, *103*, 26–40.
9. Harwood, A.R.; Revell, A.J. Interactive flow simulation using Tegra-powered mobile devices. *Advances in Engineering Software* **2018**, *115*, 363–373.
10. Teutscher, D.; Weckerle, T.; Öz, F.; Krause, M.J. Interactive Scientific Visualization of Fluid Flow Simulation Data Using AR Technology-Open-Source Library OpenVisFlow. *Multimodal Technologies and Interaction* **2022**, *6*.
11. Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* **2000**.
12. Krause, M.J.; Kummerländer, A.; Avis, S.J.; Kusumaatmaja, H.; Dapelo, D.; Klemens, F.; Gaedtke, M.; Hafen, N.; Mink, A.; Trunk, R.; et al. OpenLB—Open source lattice Boltzmann code. *Computers & Mathematics with Applications* **2021**, *81*, 258–288.
13. Li, J. Appendix: Chapman-Enskog Expansion in the Lattice Boltzmann Method, 2015. https://doi.org/10.48550/ARXIV.1512.02599.
14. Guo, Z.; Zheng, C.; Shi, B. Discrete lattice effects on the forcing term in the lattice Boltzmann method. *Physical review E* **2002**, *65*, 046308.
15. Nathen, P.; Gaudlitz, D.; Kratzke, J.; Krause, M. An extension of the Lattice-Boltzmann Method for simulating turbulent flows around rotating geometries of arbitrary shape. 06 2013. https://doi.org/10.2514/6.2013-2573.
16. Bhatnagar, P.L.; Gross, E.P.; Krook, M. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Phys. Rev.* **1954**, *94*, 511–525. https://doi.org/10.1103/PhysRev.94.511.

17. Nordström, J.; Nordin, N.; Henningson, D. The fringe region technique and the Fourier method used in the direct numerical simulation of spatially evolving viscous flows. *SIAM Journal on Scientific Computing* **1999**, *20*, 1365–1393.
18. Schäfer, M.; Turek, S.; Durst, F.; Krause, E.; Rannacher, R. *Benchmark computations of laminar flow around a cylinder*; Springer, 1996.
19. Kummerländer, A.; Dorn, M.; Frank, M.; Krause, M.J. Implicit propagation of directly addressed grids in lattice Boltzmann methods. *Concurrency and Computation: Practice and Experience* **2023**, *35*, e7509, [https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.7509]. https://doi.org/https://doi.org/10.1002/cpe.7509.
20. Geekbench. Geekbench Browser. https://browser.geekbench.com/, 2023. Accessed: June 22, 2023.