Interactive computer simulation in meteorology

Hendrik Schwanekamp
University Koblenz-Landau
Email: hendrikschwanekamp@gmail.com
Year: 2020

I. Introduction

Since the pioneering work by Jule Charney, Ragnar Fjörtoft and John von Neumann in 1950 [1] numerical simulation have been an important tool for meteorologists. Today they are used in everyday weather prediction as well as academical studies. With the availability of more and more powerful computers, the complexity and accuracy of those simulations increased. Most simulations are run remotely on supercomputers where data is stored in files for later analysis.

The use of remote supercomputers makes high computational power available to many scientists. They also promote a style of working, where it can take weeks to go from an experiment idea, over running a simulation, to analyzed data. We aim to decrease the time required from idea to first solution and make experiment design more efficient. To achieve this, we implement a graphical user interface for changing simulation settings and creating initial conditions. We also explore the use of GPUs to speed up the simulation itself. High end GPUs are available on a most modern supercomputers and relatively cheap models exist for workstations. Options in the code, to choose lower resolution and accuracy, can further increase the simulation speed. Finally, we provide a build in 3D visualization tool, to analyze the data while the simulation is still running. This allows simulations to be monitored and changed interactively, and enables a rapid prototyping approach to scientific simulation.

We choose the shallow water model for simplicity of implementation. It is also common to used it for testing new numerical methods in meteorology [2]. Most of the presented ideas can be generalized to a more complex three dimensional model based on the primitive equations. Depending on the studied domain, it can be advantageous to write the equations in cartesian or spherical coordinates. Different systems of equations and coordinates are discussed in section II. To solve partial differential equations – like those used in the shallow water model – numerically, they need to be discretized in space. Properties of the system are stored for multiple locations in space to approximate the continuous functions. Those are then used to calculate spatial derivatives. Different discretization schemes are discussed in section III.

GPUs were originally developed for rendering computer graphics i.e. updating thousands of pixels at once – a fundamental parallel task requiring high memory bandwidth and computational throughput. It was quickly realized that this architecture could be used for any sort of parallel computation. The low energy consumption per FLOP[1] relative to other processor architectures makes them a common part of modern supercomputers, with future systems planned to use them even more extensively[2].

We present our CUDA[3] implementation in section IV and show the user interface as well as interactive visualization in section V. A conclusion and ideas for future work (section VII) complete the paper.

[1]FLOP = floating point operation

II. Equations and coordinates

Here we quickly summarize different equations and coordinate systems that were considered for our simulation. For an in depth discussion and derivations of the individual equations the reader is referred to [2]

A. Primitive Equations

In computer graphics, air is commonly modeled as an incompressible Fluids. This would be an oversimplification for atmospheric simulations in meteorology, since we are interested in observing zones of different pressure and density. The equations are derived from the Euler equation for compressible fluids instead. Terms for gravity and Coriolis force are added, before a number of assumptions is made to improve numerical stability and performance of simulations. This yields the *Primitive Equations* which are used to simulate the atmosphere in three dimensions.

B. Shallow Water Equations

By assuming density and horizontal speed to be constant in the vertical direction, a two dimensional model can be derived. The state of the fluid is then described by the horizontal velocity vector (horizontal wind vector) and the height of the fluids surface. Those assumptions are useful, as long as the horizontal size of the studied fluid is much greater than its height. Hence the name *Shallow Water Equations*. The equations read [2]:

$$\frac{\partial \boldsymbol{V}}{\partial t} = -\left(f+\zeta\right)\boldsymbol{k}\times\boldsymbol{V} - \nabla\left(\Phi + K\right) \tag{1}$$

$$\frac{\partial \Phi}{\partial t} = -\nabla\cdot\left[\left(\Phi-\Phi_s\right)\boldsymbol{V}\right] \tag{2}$$

$$\zeta = \boldsymbol{k}\cdot\left(\nabla\times\boldsymbol{V}\right) \tag{3}$$

$$K = \frac{u^2+v^2}{2} \tag{4}$$

[2]For example, press releases state the planned "El Capitan" supercomputer will have GPU to CPU ratio of 4:1 https://www.extremetech.com/computing/307004-amd-hpe-el-capitan-2-exaflop-supercomputer-epyc-radeon-instinct

[3]CUDA is a framework which enables use of NVIDIA GPUs for general computing.

Here $\boldsymbol{V} = (u, v)$ denotes the horizontal velocity vector and Φ the geopotential (height times gravitational acceleration). Φ_s is the geopotential of the terrain, often simply assumed to be zero everywhere. ζ is the Vorticity – the horizontal component of the curl – and K the kinetic energy. $f = 2 * \Omega \sin(\phi)$ is called the Coriolis parameter, computed at latitude ϕ using the earth angular velocity Ω. $\boldsymbol{K}$ represents the local vertical unit vector.

Equation 2 therefore expresses the change of velocity. It takes into account the acceleration by Coriolis force (first term) and by the negative gradient of Φ (second term). The latter accelerates fluid from high pressure regions towards low pressure regions. As mass is transported by the velocity field, height of the fluid and thereby geopontential changes. Equation 3 calculates that change by taking the negative divergence of geopotential and velocity.

While limited to two dimensions, the shallow water equations are still able to model Rossby and Inertia-Gravity Waves. They are widely used test numerical methods in meteorology.

C. Spherical coordinates

Computer graphics commonly uses Cartesian coordinates. When the atmosphere of an entire planet needs to be simulated, it can be advantageous to work in spherical coordinates. Locations are then defined by longitude λ and latitude ϕ, which translates to Cartesian coordinates as explained in [3]. Special care must be taken when working with vector fields, as unit vectors in spherical coordinates change with position. This also results in a changing definitions for the differential operators (expressions involving ∇).

III. Discretization

A. Numerical methods

Multiple methods exist to solve partial differential equations numerically. All require variables to be stored at discrete locations in space to approximate the continuous functions. This is either done on an Euler-grid or using Lagrange-particles. We follow with an (incomplete) list of methods that we considered for our implementation. All but the first employ an Euler-grid.

- *Smoothed Particle Hydrodynamics* (SPH), a method based on Lagrangian particles [4], [5]. It can simulate compressible as well as incompressible fluids and has been used in computer graphics and scientific simulation. GPU implementations are documented. Due to particle movement, resolution naturally increases in high density areas and decreases in low density areas. In Atmospheric simulations that would require regular redistribution of the particles.
- *Finite Difference Method*. Spatial derivatives are computed by comparing the value of the current cell to those of neighboring cells. Extensively used in most fields that deal with fluid simulation.
- *Spectral Methods* are widely used for atmospheric simulation [2]. In spherical coordinates, they apply *Spherical Harmonics* (similar to Fourier-transformation) on the entire domain to reduce the partial differential equations into a system of ordinary differential equations. The global application of spherical harmonics makes them difficult to implement on a GPU.
- The *Finite Element Method* follows a similar concept to the the Spectral Method, but instead of a global transformation, functions with compact support are used [2]. Was successfully applied in computer graphics as well as meteorology (eg. [6]). Might be worth considering for future experiments.

For our implementation we chose the Finite Differences Method mostly for it's simplicity and easy parallelization (see section IV). Literature is available from both the computer graphics community as well as meteorologists.

To approximate spatial derivatives we use the second order accurate, symmetric formulation:

$$\frac{\partial A}{\partial x} \approx \frac{A\left(x + \Delta x\right) - A\left(x - \Delta x\right)}{2\Delta x} \tag{5}$$

For integrating in time we implemented the forward Euler Method as well as the symmetric Leapfrog Method. An in depth explanation of finite differences and time integration Methods and can be found in [2]. GPU implementation is discussed in [7], [8].

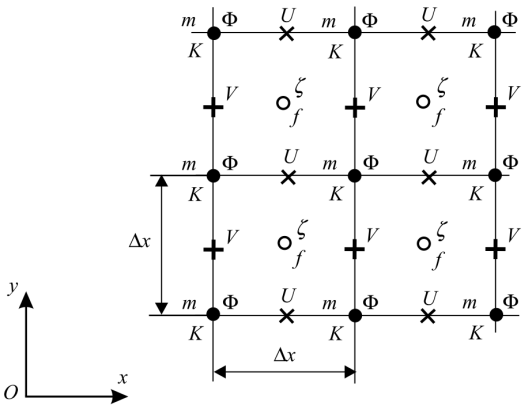


Figure 1. Positions of the variables on a C-Type grid [**?**].

B. longitude-latitude-Grid

Different types of grids can be used to store the required variables across the simulation domain. One possibility is to use a grid made up of equally sized rectangles along the coordinate axis. In Cartesian coordinates every cell occupies equal distance along the x and y axis. In Spherical coordinates cells are of equal angular size along the latitude and longitude axis. It is therefore also known as the longitude-latitude-Grid.

Data can be laid out in different ways on such a regular grid. The simplest layout stores all values at the nodes of the grid, while the more sophisticated C-Type grid stores Φ and K on the grid nodes, u and v at the center of the horizontal and vertical gridlines and ζ and f at the center of the rectangles (see figure 1). The new arrangement reduces Δx between some properties and improves the accuracy of the Finite Difference

approximation. It also simplifies the procedure to obtain the pressure gradient and the divergence of $\Phi \boldsymbol{V}$ at the correct locations [2].

C. Polar singularities

One disadvantage of having rectangles of equal angular size, is that grid cells become thinner with increasing latitude. Close to the poles cells become small enough to cause stability problems for the entire simulation. Using shorter timesteps can mitigate the problem a little, but causes the entire simulation to slow down.

The problem becomes harder to deal with directly at the poles and in the top-most and bottom-most row of the grid. Direct upper and lower neighbors are not easily defined. For the Spectral Method Merilees [9] proposes to omit the grid cell directly at the pole. He then defines a new coordinate system that allows latitudes outside of $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Coordinates can be transformed back to the original longitude-latitude-system by adding pi to the longitude and multiplying vector quantities with -1. This allows us to find the position where the neighbor of a near-polar-cell should be. Unfortunately those positions do in general not match up with any grid positions, where the required properties are actually defined. Values need to be interpolated, which is non trivial in spherical coordinates. Especially close to the poles linear interpolation in longitude-latitude space does not yield usable results [10]. This makes it difficult to use with finite differences.

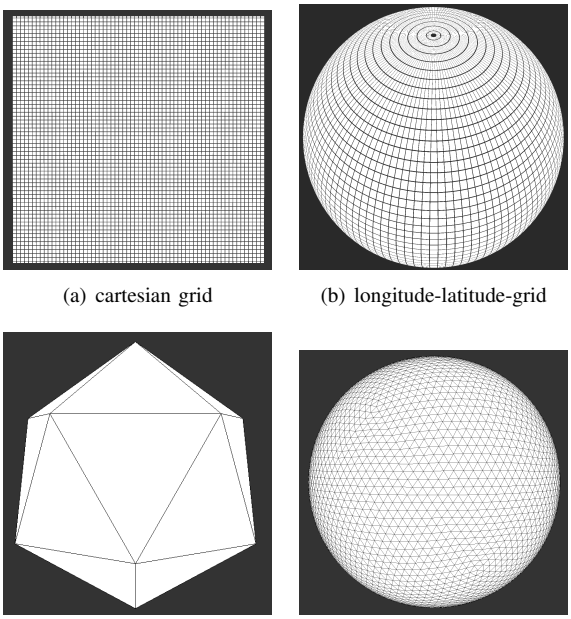
A more promising method might be to use a second rotated rectangular grid for the polar regions and merge both grids together (first proposed by Kageyama and Sato [11]).

D. Icosahedral Grid

The icosahedral grid is constructed by projecting an Icosahedron onto a sphere and subdividing the triangles until the desired resolution is achieved. The process of construction is detailed in [12]. This yields a triangle mesh of similar sized triangles in all parts of the sphere, achieving more regular discretization then the longitude-latitude-Grid. While it solves the problems at the poles more elegantly, it comes with its own set of challenges. One of those is the already mentioned construction algorithm. In addition most of the grid nodes have six neighbors, while the original corners of the Icosahedron only have five. Grid lines do not follow axis of the coordinate system which makes equation 5 unusable. A different finite differences formulation must be derived [13]. Finding the grid nodes of the triangle surrounding a specific point on the sphere (eg. for doing interpolation) does usually require the traversal of a search tree.

IV. Implementation

We implemented the Finite Difference Method on the longitude-latitude, C-Style grid for solving the shallow water equations in Cartesian and spherical coordinates. At the poles we employ Merilees algorithm (see section III-C). However our code does not seem to handle the poles correctly (see section VI).



(a) cartesian grid (b) longitude-latitude-grid

(c) base icosahedron (d) subdiveided icosahedron

Figure 2. Different grid types implement in our code.

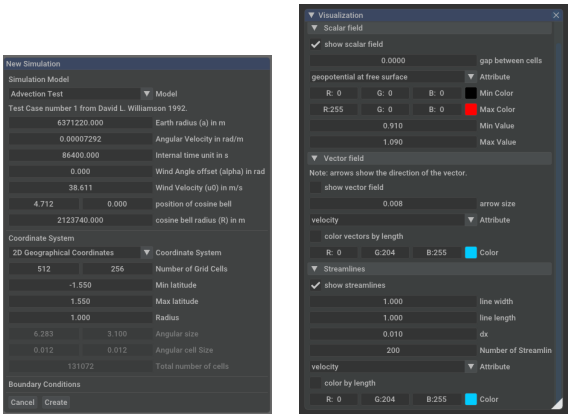
To parallelize the simulation on the GPU we use double buffering for the data on the grid. We create one GPU thread for each grid node. Each thread reads required data for its own and neighboring cells from the *read-buffer*. It then solves equation 2 and 3, using equation 5 to calculate spatial derivatives. Then velocity and geopotential are integrated over time and the new values are stored in the *write-buffer*. After all threads finish execution, read and write buffers are swapped and the computation of for next timestep can begin. The read buffer is available to the visualization module at any time during the process, so the progress can be monitored interactively (see section V).

When using a C-Grid, the calculation of the acceleration by Coriolis force as well as the gradient of $\Phi + K$ requires not only data from direct neighboring cells, but also from neighbors of neighboring cells. To cut down on the amount of memory access required, we split the algorithm in two passes. In the first pass the geopotential is updated and vorticity, Coriolis parameter and kinetic energy are computed. Those are then read by the second pass to update the velocity.

We also experimented with implementing an Icosahedral Grid. A parallel construction algorithm as well as simple visualization features are complete. Solving partial differential equations on such a grid is however more complicated and could not be implemented before the end of the projects timeframe.

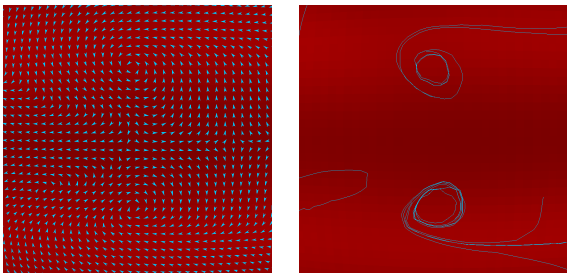
V. User Interface and Visualization

Our software uses a graphical user interface for setting up simulations. That includes selecting between different implemented models, coordinate systems and initial conditions, changing settings and monitoring performance. Some settings can even be changed while the simulation is already running. Figure 3 show example section from our user interface.



(a) creating a new simulation (b) configuring the visualization

Figure 3. The user interface allows to configure new simulations (s). The visualization settings can be changed at any time during the simulation (b)



(a) arrows (b) streamlines

Figure 4. Vector fields can be visualized by drawing small arrows or streamlines on top of the scalar field.

The user interface also provides control over the interactive 3D visualization. On the latitude longitude grid, scalar fields are visualized by rectangles whose color is calculated from a simple transfer function. Vector fields are drawn on top of that using small arrows in the center of each box. Alternatively streamlines can be shown, as seen in figure 4. Internally this is archived in a geometry shader, by tracing particles as they are advected by the vector field from a random starting position. As mentioned in section III-D finding the surrounding cell on an icosahedral grid is non trivial, therefore streamlines are only available on the longitude-latitude-grid

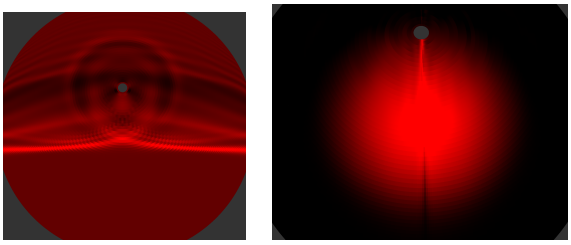
VI. Results

A. Performance

Without further optimization the two simulation allows us to update about $35.6 \cdot 10^6$ grid cells per second on a *NVIDIA GTX 1060 Mobile* commercial grade GPU, while simultaneously rendering the interactive visualization.

B. Simulation

To test our model we implemented some of the standard test-cases from [14]. In addition we performed simulations where a Gaussian over-density is added in the beginning of the simulation. Without proper boundary handling at the poles, waves are reflected, leading to unnatural results as seen in figure 5-a. The Boundary handling as proposed by Merilees (section III-C) behaves better, but distorts the advected profile significantly (figure 5-b).



(a) no pole handling (b) pole handling after Merilees

Figure 5. In the left figure waves are reflected as they hit the pole. On the right a cosine bell is advected over the pole by a constant velocity field (top to bottom) and gets distorted.

VII. Conclusion and Future Work

After considering similarities and differences of fluid simulation in meteorology and computer graphics, we compared methods used in both fields to solve partial differential equations. We presented our own implementation, using finite differences to solve the shallow water equations on a GPU. The implementation is able to process up to $35.6 \cdot 10^6$ grid cells per second on a *NVIDIA GTX 1060 Mobile* consumer grade GPU, while simultaneously visualizing the ongoing simulation in 3D.

While calculating finite differences on a longitude-latitude-grid enabled the fast development of our software, it shows significant shortcomings in the simulation results, especially in the polar region. In the future, more sophisticated methods should be considered for implementation on the GPU, keeping the polar singularity problem in mind from the beginning.

Still, we think it is possible to run low resolution/accuracy meteorological simulations interactively on a local workstation. Also larger simulations on supercomputers could benefit from GPU acceleration, as GPUs have become a common part of those system. Shipping build in tools for simulation setup and visualization with a simulation code simplifies the workflow of scientist and may reduce the training period upon switching to that software.

We propose the development of a new flexible code for atmospheric simulation that features the necessary settings to balance computational cost and resolution/accuracy of the simulation. The code should be able to harness the power of multiple GPUs on distributed supercomputer systems for large simulations, but also allow small test runs to be performed on a local workstation with a consumer grade GPU. A build in visualization would not only allow to run small simulations interactively, but also to analyze data from simulations performed remotely. That would enable a rapid prototyping approach, where resolution and accuracy can easily be increased once a promising simulation setup has been found (if necessary at all). Both interactive test runs and final production simulations would use the same meteorological model, equations and code functions, without requiring the scientist to ever leave the provided user interface.

ACKNOWLEDGEMENTS

We want to thank Alexander Wollert for supervising the project and providing helpful guidance in every meeting, as well as Dr. Wu Zheng and Vincent Carpenter for answering questions on meteorological methods and the finite difference method respectively.

REFERENCES

[1] J. G. CHARNEY, R. FJÖRTOFT, and J. Von NEUMANN, "Numerical Integration of the Barotropic Vorticity Equation," *Tellus*, vol. 2, no. 4, pp. 237–254, 1950. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2153-3490.1950.tb00336.x

[2] J. Coiffier, *Fundamentals of numerical weather prediction*. Cambridge University Press, 2011.

[3] wolfram mathworld, "Spherical coordinates," https://mathworld.wolfram.com/SphericalCoordinates.html, accessed: 2020-05-03.

[4] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner, "Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids Dan," 2019. [Online]. Available: https://interactivecomputergraphics.github.io/SPH-Tutorial/

[5] J. J. Monaghan, "Smoothed particle hydrodynamics," *Reports on Progress in Physics*, vol. 68, no. 8, pp. 1703–1759, 2005.

[6] J. Thuburn and C. J. Cotter, "A primal-dual mimetic finite element scheme for the rotating shallow water equations on polygonal spherical meshes," *Journal of Computational Physics*, vol. 290, pp. 274–297, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2015.02.045

[7] M. J. Harris, "Fast fluid dynamics simulation on the gpu." *SIGGRAPH Courses*, vol. 220, no. 10.1145, pp. 1 198 555–1 198 790, 2005.

[8] K. Crane, I. Llamas, and S. Tariq, "Real-time simulation and rendering of 3d fluids," *GPU gems*, vol. 3, no. 1, 2007.

[9] P. E. Merilees, "The pseudospectral approximation applied to the shallow water equations on a sphere," *Atmosphere*, vol. 11, no. 1, pp. 13–20, 1973.

[10] M. F. Carfora, "Semi-Lagrangian advection on a spherical geodesic grid Maria," *International Journal for Numerical Methods in Fluids*, 2007.

[11] A. Kageyama and T. Sato, ""yin-yang grid": An overset grid in spherical geometry," *Geochemistry, Geophysics, Geosystems*, vol. 5, no. 9, 2004.

[12] R. SADOURNY, A. ARAKAWA, and Y. MINTZ, " Integration of the Nondivergent Barotropic Vorticity Equation With an Icosahedral-Hexagonal Grid for the Sphere 1 ," *Monthly Weather Review*, vol. 96, no. 6, pp. 351–356, 1968.

[13] J. Thuburn, "A PV-based shallow-water model on a hexagonal-icosahedral grid," *Monthly Weather Review*, vol. 125, no. 9, pp. 2328–2347, 1997.

[14] D. L. Williamson, "A standard test set for numerical approximations to the shallow water equations in spherical geometry," *Journal of Computational Physics*, vol. 102, no. 1, pp. 211–224, 1992.