Geometry evolution in CFD simulations

Project report
Master Practical Course "CFD Lab"

Group A
Olga Glogowska, Qunsheng Huang, Tommaso Bianucci
Fakultät für Informatik
Technische Universität München

I. Idea

This Computational Fluid Dynamics (CFD) project attempts to optimize geometry via CFD output data.

This idea is based off adjoint equation geometry optimization, in which the sensitivities of drag to changes in geometry are used to optimize geometry [1], [2]. This idea was simplified and used to extend the 2D solver implemented in the CFD lab.

II. Goal

The goal is to extend the current solver implementation, allowing it to:

- Detect regions in the steady-state output that are undesirable.
- Update geometry and remove forbidden geometry.
- Repeat until convergence or until a given step.

III. Implementation

A. High-Level Implementation

Several steps were required to implement the geometry solver — a skeleton of the final code can be seen in Listing 1.

The general optimization workflow can be seen in the for loop added to the main program that would run to an arbitrary convergence criteria.

During each step of the loop, the solver, `performSimulation()`, runs until completion. The results are passed to the two functions `fillVortices()` and `updatePGM()`, which update the geometry. Finally, the function `fixGeometry()` is called to fix forbidden geometries.

```
initializeMatrices(...);
for (PGMConvergenceCriteria)
{
  performSimulation(...);
  fillVortices(..., Flags);
  updatePGM(..., imax, jmax, U, V, P, Flags);
  fixGeometry(..., imax, jmax, Flags);
  decodeAndWrite_pgm(..., PGM, Flags);
}
freeMatrices(...);
cleanup(...);
```

Listing 1. High-level skeleton of the final code.

Of all the changes to the code, the crux of the optimization is the material modification methodologies. Three are implemented in the code, namely: vortex identification, material removal and material addition.

B. Vortex Identification

In order to optimize the flow, vortexes are undesirable. The `fillVortex()` function is implemented to iteratively identify regions of re-circulation and flip the from fluid cells to obstacle cells. The concept is based on a Masters Thesis by V. Holmen [3]. A first sweep of the entire geometry is performed to identify vortex centers. The signum of velocities at localized regions in the grid are identified and if the calculated values indicate a swirl, as seen in Figure 1, a vortex center is present.

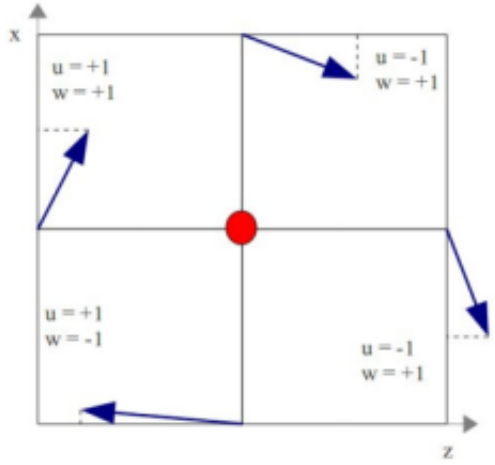

Figure 1. How to evaluate if a cell is a vortex center.

Once vortex centers are identified, multiple sweeps of the geometry are performed to mark vortex-filled regions by calculating the signum of velocities of cells adjacent to vortex cells. The locations where vortexes are identified are flipped to obstacle cells.

The initial implementation of this function was overly aggressive and targeted small vortexes throughout the flow. This often resulted in the entire flow simulation being flipped to solid. In order to prevent this, vortex identification is only

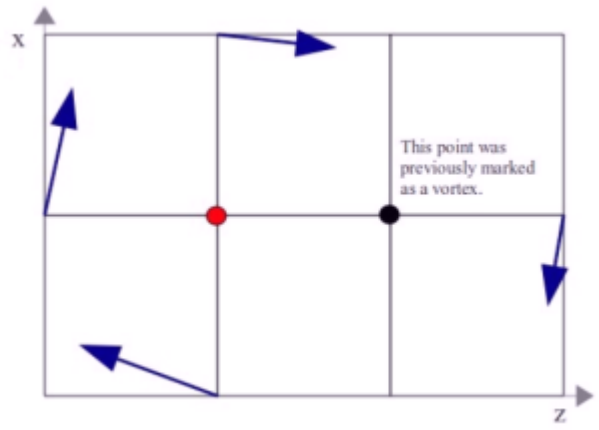

Figure 2. How vortices can be extended.

applied every few geometry update iterations (defaulting to ten in our final version). Similarly, the vortex identification would ignore vortexes smaller than a set size.

C. Material Addition

Another idea to remove unwanted fluid cells is to fill in regions with very low flow magnitudes or "dead zones" in the fluid flow. This was easily executed but constant threshold values were very sensitive to initial conditions. In order to prevent uncontrolled material addition, the final implementation uses a percentage of maximum velocity as a threshold value. Material addition was also limited to cells adjacent to existing obstacle cells.

Another problem faced was the tendency for material on the downstream-facing side of the geometry to aggregate material. This was unsurprising as it mimics sedimentation aggregation on riverbeds but it was also undesirable. A check was implemented to determine if obstacle cells faced the outflow or inflow direction and an additional flag was added to the configuration file to allow penalizing material addition to downstream-facing regions, which improved performance significantly.

D. Material Removal

The final modification method is the removal of unwanted obstacle cells. Two ideas were implemented: removal of cells at corners and removal of cells at regions of high flow. Cells were flipped from obstacle cells to fluid cells if the pressure difference across the fluid-adjacent faces was greater than set percentage. Similarly, cells would be flipped from solid to fluid if the adjacent fluid cells had velocity magnitudes greater than a preset value.

Masking of certain geometries is also implemented — allowing for a separate PGM to indicate regions that should not be changed when updating geometries. Masks are typically applied to inflow and outflow regions. Can also be used to prevent geometries from "floating" downstream due to repeated material additions and removals.

IV. Analysis and Results

Once these methods were implemented and the input parameters were tuned, we were surprised to see relatively positive results. While it is clear by visualizing the results that the final geometries are not optimal, the results seem to converge to a maximum value. The outflow (calculated by integrating out-facing velocities at outflow cells) was seen to improve with the implemented geometry optimization. Please view the attached videos for the various geometry test cases implemented: flow over step, shifted flow over step, obstacle, simple bend and simple channel.

V. Further Improvements

Based on the limited amount of allotted time, the final product achieves most, if not all, of the set objectives. However, while the specific test cases provided in our code work relatively well, the code can still be significantly improved. There are three main regions which can be targeted for improvement: convergence criterion, automation of parameter tuning and parallelization.

A. Convergence Criterion

In order to avoid excessive wait times, a break criterion was implemented such that the CFD solver would exit when SOR iterations fell below a set limit. This method is not extremely robust and can result in undesirable material addition. Additionally, we were unable to design a robust method to stop geometry modification. In the final product, we end the simulation after a preset number of PGM update iterations. Finding an algorithm to perform this robustly would greatly improve the implementation.

B. Automation of Parameter Tuning

There are many, many variables added to the configuration file, each of which significant effects on the geometry updates. Ideally, we could automatically tune these variables for each test case. However, this is beyond the scope with the given time.

C. Parallelization

Most regions, if not all, of the code are fully parallelizable (Jacobian iterations through the various matrices). It would not be a difficult task to implement MPI routines for greater speedups. This is especially the case for higher resolution figures, which significantly increase wait times for results.

References

[1] C. Othmer, "Adjoint methods for car aerodynamics," *Journal of Mathematics in Industry*, vol. 4, no. 1, p. 6, 2014.

[2] C. Othmer and T. Grahs, "Cfd topology and shape optimization with adjoint methods," *VDI BERICHTE*, vol. 1967, no. 1, p. 61, 2006.

[3] V. Holmén, "Methods for vortex identification," *Master's Theses in Mathematical Sciences*, 2012.