Narrow Band FLIP for Liquid Simulations: Implementation and Optimization

BAPTISTE BOULAN, Telecom Paris

Liquid simulations using the Fluid Implicit Particle (FLIP) method provide high-quality visual results but are traditionally computationally expensive due to the need for a dense sampling of particles throughout the entire fluid volume. This report details the implementation of a "Narrow Band" FLIP approach, which restricts Lagrangian particles to a thin layer near the liquid surface while representing the interior on a regular Eulerian grid. I describe the transition from a 2D CPU-based prototype to a fully 3D GPU-accelerated simulation capable of handling up to 18 million particles. Our results demonstrate a 2x speedup in stable scenarios, though challenges remains regarding visual inertia and incompressibility at the narrow band interface.

Additional Key Words and Phrases: Fluid simulation, FLIP, Narrow Band, GPU Acceleration, Level Set

1 INTRODUCTION

The Fluid Implicit Particle (FLIP) method is a staple in computer graphics for producing non-dissipative, high-frequency liquid details such as splashes and thin sheets [1]. However, the standard approach requires filling the entire fluid volume with particles, most of which contribute little to the visual surface.

In this project, we implement the "Narrow Band FLIP" (NB-FLIP) algorithm as proposed by [1]. The core idea is to maintain particles only within a distance R of the surface. This report chronicles the development process, from basic solvers to massive-scale GPU parallelization.

2 BASE IMPLEMENTATION

2.1 2D Foundation and Solvers

The project began with a 2D implementation of the FLIP/PIC algorithm to establish a baseline. A critical component of any grid-based fluid solver is the pressure projection step, which ensures the velocity field is divergence-free.

Initially, a **Gauss-Seidel** solver was implemented. However, it yielded poor results: the convergence was too slow to maintain incompressibility, leading to visible volume loss and "squishy" fluid behavior (see Figure 1).

To resolve this, I transitioned to a **Conjugate Gradient (CG)** solver. CG is a more robust iterative method for sparse, symmetric positive-definite systems, providing much faster convergence and stable incompressibility for the pressure Poisson equation. It immediately gave better results, showing realistic behavior and having the residual converging to 10^{-7} in about 60 iterations, whereas the Gauss-Seidel method was struggling to get under 10^{-4} in 100 iterations (even worse with Jacobi's method).

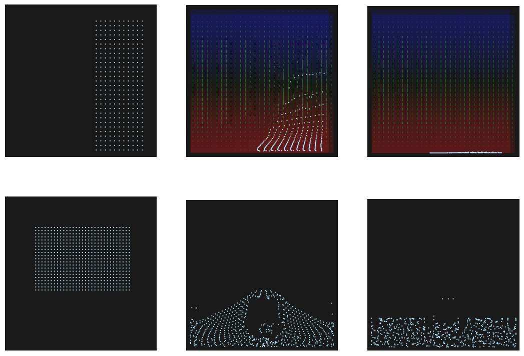
Fig. 1. Comparison of Gauss-Seidel (top) results showing poor incompressibility versus the stable Conjugate Gradient (bottom) output.

2.2 GPU Acceleration

Scaling the simulation beyond a few thousand particles on the CPU proved infeasible due to computational constraints. To achieve high-performance rendering, the entire simulation logic was migrated to the **GPU**. A critical challenge was avoiding the PCIe bottleneck: transferring particle data, grid data, velocities, and matrices back and forth between the GPU and CPU for every frame—or worse, for every iteration—would have drastically slowed down the algorithm. This was the exact opposite of the intended performance optimization.

Another significant hurdle was managing concurrent writes to shared buffers. For example, in the P2G (Particle-to-Grid) shader, which executes for every particle, multiple particles often needed to update the same grid velocity simultaneously. To resolve this, atomic operations such as **atomic adds** and **compare-and-swap (CAS)** were required. However, these operations are not natively supported in GLSL for floating-point values, adding another layer of complexity to the implementation.

Despite these challenges, the optimization was successful. By keeping the entire simulation loop—including P2G mapping, pressure solving, G2P mapping, and advection—on the GPU, the algorithm's performance improved dramatically. This allowed for real-time simulation of approximately **500,000 particles** at around 50 frames per second, a 100-fold increase from the CPU-limited 5,000 particles.

This report is submitted as a part of project for Advanced 3D Computer Graphics (IMA904/IG3DA), Telecom Paris.

The original work is introduced by [1].

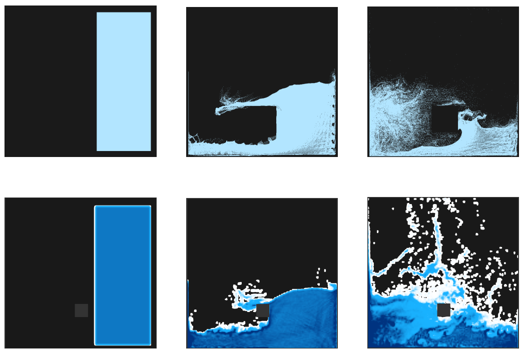

Fig. 2. Conjugate gradient results with GPU: 500,000 particles at 50 FPS

3 3D EXTENSION AND NARROW BAND

3.1 Scaling to 3D

With the GPU infrastructure already in place, the simulation was extended to 3D. While the underlying logic remained conceptually similar, managing 3D grid indexing and parallel neighbor lookups required meticulous implementation to sustain performance.

However, the massive increase in grid size—from a 2D grid of 128×128 (16,384 cells) to a 3D grid of 64×64×64 (262,144 cells)—significantly degraded performance. This highlighted the critical importance of migrating to GPU **before** scaling to 3D, to avoid major bottlenecks.

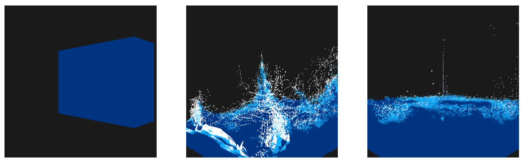

Fig. 3. 3D FLIP/PIC simulation: 500,000 particles at 7 FPS

3.2 Level Set Computation

The Narrow Band FLIP algorithm relies on a level set function, ϕ, to track the signed distance to the liquid surface. While the original paper [1] suggests initializing ϕ using an analytical distance with quick passes over the grid and then advecting ϕ before correcting it with a particle-based ϕ^p, this approach resulted in a constant gradient of $-h$ in our implementation, which proved insufficient for stable simulations.

To address this, we developed a custom Dijkstra-like iterative algorithm for level set computation. The algorithm initializes air cells using the particle level set and propagates distances into the fluid interior. This method yields smoother distance fields, as illustrated in Figure 4, though occasional instabilities may still arise due to the discrete nature of grid-based propagation.

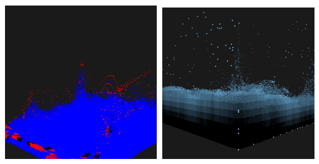

Fig. 4. Comparison of level set initialization methods: (left) the paper's approach, which produces a binary-like field, and (right) our Dijkstra-like algorithm, which generates a smooth gradient.

3.3 Cull and Resample

The core optimization of the Narrow Band FLIP method lies in the *Cull and Resample* step. This step ensures that particles are maintained only within a narrow band around the liquid surface, significantly reducing computational overhead. The process involves two key operations:

Culling: Particles outside the narrow band threshold are removed. Resampling: New particles are seeded in cells near the surface where the particle density falls below a predefined minimum. The pseudocode for this process is provided in Listing 1. The threshold for the narrow band is set to 5h5h5h, where hhh is the grid cell width. Particles are culled if their distance to the surface exceeds this threshold. Resampling is performed in cells where the level set value indicates proximity to the surface, but the particle count is below the minimum required density.

Listing 1. Cull and Resample logic (pseudo C++ code)

```
void Simulation::cullAndResample() {
    // ...Retrieve data from GPU...
    float threshold = 5.0f * h;
    std::vector<Particle> keptParticles;
    std::vector<int> cellCounts(grid.total_cells, 0);
    Copier

    // Cull particles outside the narrow band
    for (const auto& p : particles) {
        int idx = grid.gridIdx(p);
        if (grid.finalLevelSet[idx] <= threshold) {
            keptParticles.push_back(p);
            cellCounts[idx]++;
        }
    }

    // Resample sparse cells near the surface
    for (int idx = 0; idx < grid.total_cells; ++idx) {
        if (grid.levelSet[idx] > threshold - 2*h &&
            grid.levelSet[idx] < threshold &&
            cellCounts[idx] < minCellCount) {
            // Seed new particles to
            // maintain minimum density
            int pToAdd = minCellCount - cellCounts[idx];
            for (int i = 0; i < pToAdd; ++i) {
                Particle newP = grid.seedParticle(idx);
                keptParticles.push_back(newParticle);
            }
        }
    }
    particles = std::move(keptParticles);

}
```

In this implementation, the grid.seedParticle(idx) function generates a new particle within the cell idx, ensuring that the particle distribution remains uniform and consistent with the narrow band requirements. The choice of $5h$ as the threshold,instead of $3h$ as mentionned in the paper, balances computational efficiency with the need to capture surface details accurately.

4 PERFORMANCE AND RESULTS

The NB-FLIP implementation achieved a **2x speedup** in stable scenarios where the fluid occupies a large volume. In highly turbulent scenes where the narrow band effectively covers the entire fluid, performance was comparable to standard FLIP (see Figure **??**). To handle the massive particle counts (up to **18 million** on a 128^3 grid), we replaced individual particle rendering with a **Marching Cubes** surface extraction.

In the standard FLIP, the time to compute one frame is almost consistently *, whereas the the NB-FLIP, even though it still is $2.5s$ at the beginning, when the liquid settle, the computational time is reduced to about a second, which is about the time gain that the paper was promising.

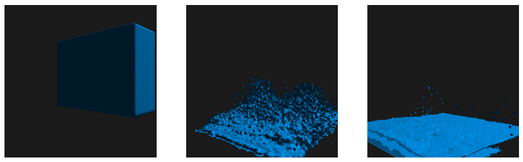

Fig. 5. Final version, with grid size 128^3 for about 18 millions particles with marching cubes.

5 LIMITATIONS AND FUTURE WORK

While efficient, the current implementation exhibits less "organic" behavior than the original 2D code, specifically a loss of inertia and occasional incompressibility artifacts at the band boundary. Future work will focus on implementing a more accurate level set and exploring surface tension to improve the physical realism.

REFERENCES

[1] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. 2016. Narrow Band FLIP for Liquid Simulations. *Computer Graphics Forum* 35, 2 (2016), 225–232.