GPU-Accelerated Particle-Based Fluid Simulation

Ivan Mix [*1] **Jacob Dudik** [*1] **Abhinav Vemulapalli** [*1] **Nikola Rogers** [*1]

Abstract

This paper details the implementation of a high-performance Smoothed Particle Hydrodynamics (SPH) solver utilizing NVIDIA's CUDA API. By exploiting the massive parallelism of modern GPU architectures, our implementation addresses the computational bottlenecks inherent in traditional CPU-based fluid dynamics. The solver numerically models fluid phenomena through the governing equations of continuity and momentum, utilizing specific smoothing kernels for density and pressure calculations. Visualization is achieved via a seamless CUDA-OpenGL interop pipeline, enabling real-time interactive rendering of the fluid field.

1. Introduction

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian, mesh-free method used to simulate fluid flows. Originally developed for astrophysics (Gingold & Monaghan, 1977; Lucy, 1977), SPH represents the fluid as a set of discrete particles that carry physical properties such as mass, position, velocity, and density. This approach is particularly effective for simulating complex free-surface flows, splashing, and highly deformable boundaries, which are computationally expensive to track with fixed meshes.

However, the computational cost of SPH scales poorly on sequential processors. The interaction forces between particles require searching for neighbors within a smoothing radius, a process that naively scales at $O(N^2)$. To achieve real-time performance for simulations involving thousands of particles, hardware acceleration is necessary (Harada et al., 2007).

This project implements a GPU-accelerated SPH solver using CUDA C++. We focus on optimizing memory throughput and minimizing data transfer latency between the simulation (Compute) and visualization (Graphics) stages.

2. Methodology

The simulation approximates the Navier-Stokes equations for incompressible flow. The state of the fluid is updated in discrete time steps Δt by computing the forces acting on each particle i from its neighbors j.

2.1. Smoothing Kernels

The SPH method interpolates quantities at a location $\mathbf{r}$ using a weighting function $W(\mathbf{r}, h)$, where h is the smoothing length. Our implementation utilizes two specific kernels optimized for stability and force computation, as defined in 'sph_kernels.cu' (Müller et al., 2003).

Poly6 Kernel (Density Estimation): For density estimation, we use the Poly6 kernel. It is chosen for its mathematical simplicity and efficiency in calculating the scalar density field. This kernel is implemented in the device function `poly6` and is used to compute the density.

Spiky Kernel (Force Calculation): For pressure and viscosity forces, the gradient of the kernel is required. The Poly6 kernel has a vanishing gradient as $r \to 0$, which causes particle clustering. To mitigate this, we employ the Spiky kernel, which creates a non-zero repulsive force as particles approach each other. This is implemented in the device function `spiky` within our CUDA code.

2.2. Governing Equations

The physics simulation executes via a two-stage pipeline on the GPU. The initial kernel, `compute_density_pressure`, iterates through the particle set to evaluate the local density ρ_i. To approximate incompressibility, the solver then derives the pressure p_i via the Tait equation of state (Becker & Teschner, 2007), utilizing the user-defined stiffness and rest density parameters. With the pressure field established, the subsequent kernel, `compute_forces_and_integrate`, resolves the net acceleration. This stage accumulates the external gravity vector and computes the pressure gradient forces, ensuring symmetry to satisfy Newton's third law of motion.

3. Implementation Details

3.1. CUDA Architecture

The simulation maintains state in a unified `Particle` struct containing `float3` vectors for position and velocity, alongside scalar density and pressure values. We utilized an Array of Structures (AoS) pattern for particle data. While Structure of Arrays (SoA) often yields higher memory throughput on GPUs, AoS was selected here to reduce code complexity within the interaction kernels, balancing development time against raw performance.

The physics simulation relies on a continuous three-stage kernel pipeline per time step. The process begins with `compute_density_pressure`, which evaluates local particle density using the Poly6 smoothing kernel and subsequently derives pressure via the Tait equation. These properties feed into the primary integration kernel, `compute_forces_and_integrate`, which resolves the net acceleration from pressure gradients, gravity, and boundary constraints. This kernel also manages interactive dynamics, applying repulsive forces from the virtual stirring rod. The pipeline concludes with a parallel reduction step, `reduce_vmin_vmax`, employing atomic Compare-and-Swap (CAS) operations to efficiently determine global velocity bounds for the dynamic gradient visualization.

3.2. Graphics Interoperability

A critical optimization in this solver is the Zero-Copy visualization pipeline, designed to circumvent the bandwidth bottleneck inherent in traditional host-side data transfers. By registering the OpenGL Vertex Buffer Object (VBO) via `cudaGraphicsGLRegisterBuffer`, the simulation grants CUDA direct write access to the graphics memory. Per frame, these resources are mapped into the CUDA address space using `cudaGraphicsMapResources`, permitting the kernel `update_render_buffer_compact` to populate position and velocity data directly in place. This allows OpenGL to perform instanced rendering immediately without CPU intervention. In scenarios where hardware or driver constraints prevent this direct linking—such as certain WSL2 environments—the system automatically degrades to a host-round-trip buffer strategy implemented in `stepSimulationFallback`.

3.3. Interactive Controls

The application integrates `Dear ImGui` to provide real-time control over simulation parameters. Users can dynamically adjust the gravity vector, stiffness constant k, and wall damping factors. Additionally, a "stirring rod" interaction is implemented using ray-plane intersection, allowing users to apply repulsive forces to the fluid using the mouse.

3.4. Visual Quality

The simulation renders particles as 3D spheres using a custom shader. We implemented two color modes found in ‘shaders.cpp‘:

- **Plasma:** A gradient from blue to purple/yellow, highlighting high-velocity regions.
- **Ocean:** A gradient from deep navy to white, simulating typical fluid colors like water.

The visualization successfully depicts fluid sloshing, settling, and interaction with the cylindrical stirring rod. These shaders automatically contrast the colors based on the min and max particle velocities, so an objectively slow particle may appear bright because it is the fastest relative to all other particles in the fluid.

4. System Interface and Configuration

After launching the simulator with `vglrun -d egl ./simple_sph`, you'll be presented with a screen to configure the initial parameters for the simulation. The different parameters and explanations are as follows:

Table 1. Simulation Configuration Parameters.

Category	Parameter	Description
Core	Particles	Total particle count initialized.
	Time Step (Δt)	Integration step. Smaller steps improve stability.
	Visual Size	Sphere rendering radius (visual only).
Physics	Smooth Rad (h)	Interaction cutoff. High cost ($O(N^2)$).
	Mass	Mass of a single particle.
	Rest Density (ρ_0)	Target equilibrium density for pressure solver.
	Stiffness (k)	Gas constant. Controls compressibility (water-like vs. spongy).
Env.	Gravity	Vertical acceleration vector.
	Wall Damp	Restitution coeff. $(0 - 1)$. Controls impact energy loss.
	Box Size	Dimensions of the simulation cube.

Once the initial parameters have been chosen, the simulation will begin. Navigation is handled via left-click drag to enable rotation, and scroll wheel to adjust the zoom level. If you right-click and drag around, a "stirring rod" will appear and agitate the particles. You can change the radius of the "stirring rod" using the slider in the "Live Controls" panel. In that panel, you can also configure the gravity, stiffness, and damping parameters which will update the simulation in realtime. Lastly, you can also change the colors between gradients of blue and "plasma" depending on your goals. For more fine-grained control over the parameters, holding the Ctrl key and clicking on the number allows you to enter specific numbers.

5. Results

We evaluated the performance of the simulation on an NVIDIA GPU. The benchmarks focused on the Frame Rate (FPS) and the stability of the simulation under varying particle loads.

5.1. Performance

The table below details the performance scaling. The "Interop" column denotes the frame rate achieving using the optimized CUDA-OpenGL pipeline, while "Fallback" denotes the performance involving CPU memory copies.

The results demonstrate the critical importance of the CUDA-OpenGL Zero-Copy interop. On the L40S server configuration, enabling Interop resulted in an 8x performance increase (31 FPS vs 240 FPS) at low particle counts. Furthermore, the CPU Fallback mode on the server suffered significant performance degradation likely due to remote visualization overhead (`vglrun`), whereas the local laptop configuration maintained reasonable fallback performance.

Table 2. Simulation Performance using **GPU** (Interop).

Configuration	Particles	FPS
Config A Xeon Gold 6548Y+ Nvidia L40S 45457.6 MB GPU vRAM	1,024	240.995
	4,096	178.893
	16,384	78.089
	65,536	14.542
	100,000	7.357
Config B AMD Ryzen 9 6900HS Nvidia RTX 3070 Ti (Laptop) 8191.5 MB GPU vRAM	1,024	164.892
	4,096	123.326
	16,384	42.4837
	65,536	7.46559
	100,000	3.92858

Table 3. Simulation Performance using **CPU Fallback** (No Interop).

Configuration	Particles	FPS
Config A Xeon Gold 6548Y+ Nvidia L40S 45457.6 MB GPU vRAM	1,024	31.312
	4,096	9.087
	16,384	2.887
	65,536	0.743
	100,000	0.485
Config B AMD Ryzen 9 6900HS Nvidia RTX 3070 Ti (Laptop) 8191.5 MB GPU vRAM	1,024	128.624
	4,096	90.0645
	16,384	30.6588
	65,536	6.91873
	100,000	3.40504

6. Discussion and Future Work

The current implementation successfully demonstrates the viability of GPU-accelerated SPH. However, the $O(N^2)$ neighbor search limits scalability beyond approximately 10,000 particles.

Future optimizations planned for the roadmap include:

1. **Spatial Partitioning:** Implementing a Uniform Grid or Spatial Hash Map to reduce neighbor search complexity from $O(N^2)$ to $O(N)$.

2. **Boundary Handling:** Transitioning from the current simple box-bounds check to a particle-based boundary method (Ghost Particles) to better maintain pressure equilibrium at the walls.

3. **Advanced Rendering:** Replacing point-sprite spheres with Screen Space Fluid Rendering (SSFR) or Raytracing (NVIDIA OptiX) for realistic water surfaces.

7. Conclusion

This project delivered a fully interactive, GPU-accelerated fluid simulator. By leveraging CUDA for physics calculations and OpenGL for instanced rendering, we achieved real-time performance significantly exceeding CPU capabilities. The implementation of Zero-Copy interop proved critical for maintaining high frame rates, validating the architectural choices made during development.

8. Statement of Work

Ivan: Implemented all SPH physics kernels in CUDA. Engineered the cross-platform build system to ensure seamless functionality on both Linux and Windows environments.

Jacob: Created the OpenGL visualizer for the simulation and handled the GUI for user interaction. Documented and refactored the code base to meet standards and project requirements.

Abhinav: Assisted Ivan with the CUDA kernel code and enabled GPU Interop for faster display. Helped revise and finish the final report with collecting data. Also worked on getting the codebase working in PACE-ICE and the relevant instructions for that process.

Nikola: Designed the stirring rod interaction physics and visualization. Refactored the codebase to be more understandable and follow the proper formatting.

References

Becker, M. and Teschner, M. Weakly compressible sph for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 209–217, 2007.

Gingold, R. A. and Monaghan, J. J. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

Harada, T., Koshizuka, S., and Kawaguchi, Y. Real-time particle-based fluid simulation with rigid body interaction. In *Proceedings of the 15th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, pp. 13–18, 2007.

Lucy, L. B. A numerical approach to the testing of the fission hypothesis. *The astronomical journal*, 82:1013–1024, 1977.

Müller, M., Charypar, D., and Gross, M. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 154–159, 2003.