Harnessing Compute Shaders for the Real-Time Simulation of Physically Accurate, High-Energy Fluid Dynamics

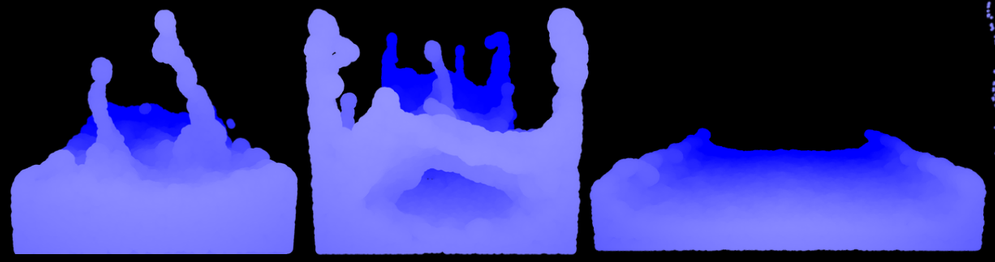
Evelyn Moore s4104592 2025

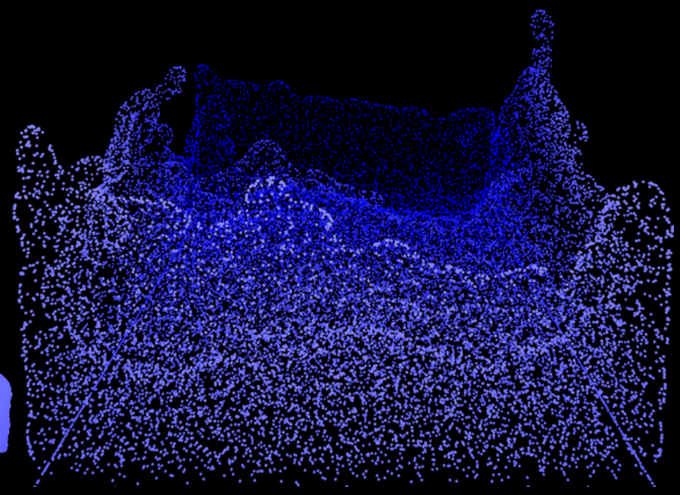
Introduction:

The purpose of this investigation is to understand and analyse the suitabilty of differing Computational Fluid Dynamics (CFD) methods for different use cases and their real-time performance when implemented using compute shaders in Unity. CFD is concerned with replacing the Navier-Stokes equations, which govern fluid flow but often lack analytical solutions, with numbers which when advanced in space and/or time describe the movement of a flow field (Anderson et al. 2013). There are two methods for fluid simulation covered here, a particle-based (Lagrangian) approach, and a grid-based (Eulerian) approach (Wang et al., 2024).

Lagrangian Implementation:

Lagrangian implementations include thousands of individual particles, each with values such as velocity, pressure, and density. Each particle interacts with those around it and influences these values according to a smoothing function, typically a Gaussian kernel (Wang et al. 2024). It is difficult to enforce compressability in a Lagrangian implementation (Huang, 2024), but this is typically not as important in applications such as video games, where accuracy is less important than performance. The Lagrangian simulation implemented appears visually striking and realistic, and with an added viscosity value could simulate many types of liquid effectively.

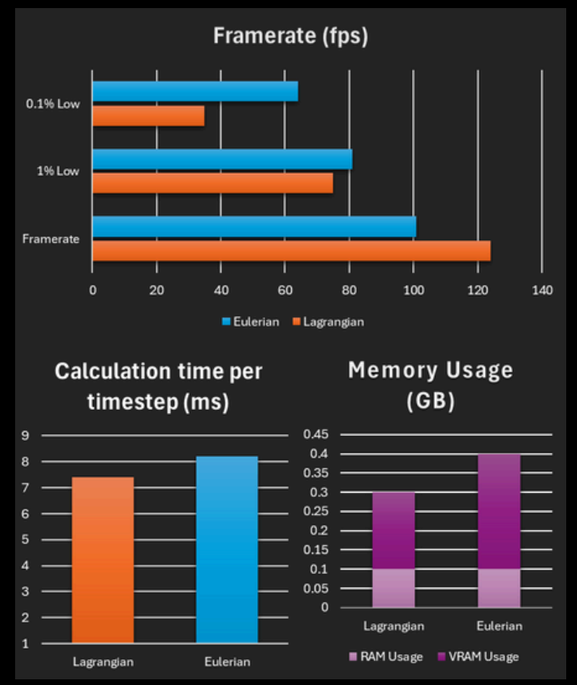




Performance:

The Lagrangian implementation performs as expected, efficiently leveraging the GPU to enable many millions of calculations per second. On average, it calculates one timestep in 7.4ms, which is enough to obtain framerates of greater than 120 fps. However, much lower dips are seen than with the Eulerian implementation (Lagrangian has 0.1% lows of 35 fps compared to Eulerian's 64 fps). This is particularly noticable while using the Lagrangian method because short dips to low framerates can cause fluid blow-up.

During testing, the Eulerian implementation appeared to use more of the CPU, and also had markedly higher memory usage. This indicates that not enough of the workload is being offloaded onto the GPU or perhaps the work being done on the GPU is not being executed efficiently. At the moment, only the diffusion calculations are performed on the GPU, and although these make up the majority of the calculations in most cases, clearly values such as grid point velocities could also be implemented as compute shaders to further increase performance. This is likely why the expected result of the Eulerian simulation being more efficient than the Lagrangian method was not obtained.

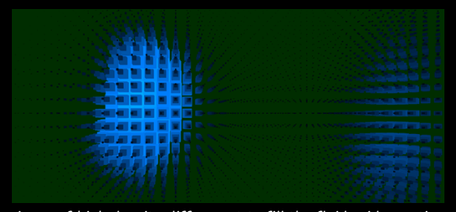


Values are an average recorded over 3 minutes of constant interaction with the simulation.

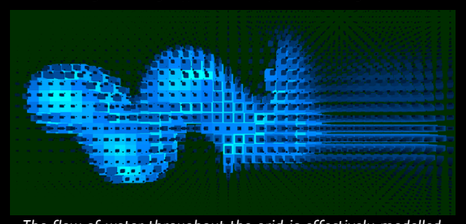
Stability:

Lagrangian is prone to liquid explosions and instability at high gravity or high timescales. It is particularly vulnerable when the computer is under load from other processes, and a single moment of reduced framerate can cause instability within the simulation.

Eulerian is prone to energy decay over time or loss/gain of density and velocity over long time periods or many interactions, partly due to floating point inaccuracies. This was seen in the simulation implemented, where very slight changes to the density decay rate would cause the bounds to completely fill with fluid or completely empty over the course of a few minutes.



Areas of high density diffuse out to fill the fluid grid over time.



The flow of water throughout the grid is effectively modelled.

Eulerian Implementation:

Eulerian implementations efficiently approximate solutions to the Navier-Stokes equations (Horváth, 2012). They typically model the fluid field as a grid of static points, each of which stores the velocity, density and pressure of the fluid at that point. The values of the fluid at points in between grid positions can be calculated simply by interpolating between the values at grid points, although this requires a fairly high density grid to not lose detail.

The Eulerian implementation effectively models fluid flow over the surface of a large body of water and the inclusion of diffusion makes it suitable for the modelling of compressible fluids such as smoke, flames, and steam, although additional kernels would need to be added to the compute shader to calculate vortices.

Conclusion:

In conclusion, Lagrangian implementations are more suited to more technical games where fluids are a core focus, as for most applications the risk of fluid blow-up is significant unless the time step is made very small, in which case the simulation becomes intensive even when calculated on the GPU. Eulerian implementations are well suited for large bodies of water, such as waves crashing against a rocky shore, or small visual effects such as smoke or steam, due to its relatively inexpensive performance. This is evidenced by the success of game engine plugins such as FluXY (Virtual Method, 2021), which use a 2D eulerian method to enable users to create effects such as oil spills, pressure waves in water, and fire.

References:

Anderson, J.D., Degrez, G., Dick, E., & Grundmann, R. (2013) p.6 *Computational Fluid Dynamics, An Introduction.* Accessed at https://books.google.co.uk/books/about/Computational_Fluid_Dynamics.html?id=PM3yCAAAQBAJ&redir_esc=y on 25/05/2025

Horváth, Z. (2012) *Real-Time Particle Simulation of Fluids.* Accessed at https://www.researchgate.net/publication/266223633_Real-time_particle_simulation_of_fluids on 27/04/2025

Huang, Y. (2024) *Variational Interference via Smoothed Particle Hydrodynamics.* arXiv. Accessed at https://arxiv.org/pdf/2407.09186 on 20/05/2025

Virtual Method (2021) *FluXY. Accessed at https://assetstore.unity.com/packages/tools/physics/fluxy-2-5d-fluid-simulator-203795* on 26/05/2025

Wang, F., Sun, Z., & Hu, X. (2024) *An efficient truncation scheme for Eulerian and total Lagrangian SPH methods.* arXiv. Accessed at https://arxiv.org/pdf/2405.05155 on 22/05/2025