Fluid Dynamics for Games: A Literature Review

Zhaorui Zhang[1,a,*], Yongzhi Zhuang[2,b], Yiqun Zhong[3,c], Bowen Chen[4,d]

[1]Northeastern University, Boston, United States
[2]Sichuan University, Sichuan, China
[3]Huazhong University of Science and Technology, Wuhan, China
[4]Communication University of China, Hainan, China
a. jaysonzr2002@gmail.com, b. nariyz@outlook.com, c. zhongyiqun0@gmail.com,
d. 202229013098N@cuc.edu.cn

*corresponding author

Abstract. Fluid simulation in video games presents significant challenges in balancing real-time performance and visual accuracy. This paper discusses developments concerned with fluid simulation techniques that optimally choose between computational efficiency and realistic fluid dynamics. Major techniques such as DCGrid, Incompressible Smoothed Particle Hydrodynamics (ISPH), Weakly Compressible SPH (WCSPH), Implicit Incompressible SPH (IISPH), and the Finite Volume Method (FVM) have been evaluated for application in several scenarios. This paper will highlight the trade-offs between accuracy and speed involved, especially in real-time simulations, and how each of these methods addresses such challenges. This paper aims at an in-depth understanding of the various fluid simulation strategies that can result in highly immersive and visually engaging gaming experiences.

Keywords: video games, Fluid Dynamics, fluid simulation strategies

1 Introduction

Fluid dynamics has become increasingly important with the development of technology in video games, where infusing fluids can create realistic, graphically beautiful game environments. Fluid effects like water, smoke, and splashes add a lot more to a game than just appealing visuals; they help the player feel immersed further into their virtual experience. Whether a character is wading through a river, waves crash onto the shore, or smoke billows from an explosion, realistic fluid simulation serves to provide more dynamic and interactive virtual worlds. However, the challenge in simulating these complex behaviors in real time faces performance constraints and often forces developers to make a trade-off between correctness and speed for every application.

Real-time fluid simulation in game development is very much a balancing act. On the one hand, there would be an approximation of high-fidelity visuals of the natural motion of fluid, while on the other hand, game engines do need to keep up with consistent performance. Fundamentally, two issues may be perceived in how to render fluid in an efficient manner without losing computational speed and how the realistic interactions between fluids and objects in a game, such as characters or terrain, are managed. Both of these are integral parts of the player's experience and need to be responsive and believable while running within the limits of real-time processing.

CONF-CIAP 2025, January 17-23, Eskisehir, Turkey
Copyright © 2025 EAI
DOI 10.4108/eai.17-1-2025.2355245

Over the years, a number of different techniques have been developed for propagating fluid simulation in games, each with its strengths determined by the demands to which the game may be put. This review will elaborate on some important methods of fluid simulation based on four papers, each with different advantages in different game scenarios. The first paper [1] mainly introduces the method DCGrid, which is a grid-based approach that automatically adapts the resolution of the grid w.r.t. the behavior of the fluid. It has smoother performance since more resources are put toward complex fluid interactions. The second [2] compares two particle-based techniques: Incompressible Smoothed Particle Hydrodynamics (ISPH) and Weakly Compressible SPH (WCSPH). Although ISPH gives higher accuracy with the help of constant fluid volume, WCSPH provides faster simulations, hence more applicable in real-time applications where speed is vital. The third paper [3] introduces an interesting thought based on the method Implicit Incompressible SPH (IISPH). It enhances the traditional IISPH in certain ways such that it increases stability and allows larger steps in time, hence enabling more complicated fluid simulations while keeping computational costs lower. Finally [4], the Finite Volume Method (FVM) divides the fluid into smaller volumes. Therefore, it is very powerful for big-scale water simulations that occur within open-world video games or places with vast bodies of water.

This review of such methods will ideally enable an understanding of how variant fluid simulation techniques can be put to work in game development and will, consequently, allow developers to select the best approach, given the requirements of a certain game. Be it computational speed or visual realism, understanding the strengths and weaknesses of these techniques is going to be crucial for effective fluid dynamics simulation in modern games.

2 Background

In the context of game development, fluid simulation plays a vital role in creating more realistic and immersive virtual environments. However, as described in the introduction, real-time fluid simulation is very challenging because of the intrinsic trade-off between visual accuracy and computational performance. The following introduces two basic viewpoints in the simulation of fluids: Eulerian and Lagrangian viewpoints, together with the explanation of the Navier-Stokes equations, the foundation of most fluid simulation methodologies [5]. It will be important to understand them in order to later on understand how most modern fluid simulation techniques are put into practice in games.

Fluid dynamics can be described from two primary perspectives: the Eulerian and Lagrangian viewpoints [5]. In the Eulerian viewpoint, attention is directed to fixed points in space, observing the change of properties of the fluid-such as velocity and pressure-with respect to time at these points. In this viewpoint, space is divided into a grid, and the fluid is followed in its motion across the grid. This typically occurs on the grid and is used in fluid simulations that call for fixed spatial references; hence, it is suitable for smoke or large water body simulations where precision is paramount.

In contrast, the Lagrangian viewpoint defines the motion of fluids by following every particle throughout their motions in space. Instead of the instantaneous properties of fluids at one stationary point, this viewpoint is interested in the trajectory of each particle. It is a particle-based method that is in common use within SPH, wherein fluids are modeled as an assemblage of interacting particles. SPH works well in the case of simulations where complicated and dynamic behaviors must be realized, including splashes and fluid-object interactions, hence allowing more flexibility in the representation of fluid motion in real-time games.

Each perspective has its advantages in game development. The Eulerian viewpoint is to be enabled on large-scale, gridbased simulations with full and accurate control over the activities of the fluid. This could be seen in FVM. On the contrary, Lagrangian viewpoint-based SPH techniques, such as WCSPH and IISPH, have much to offer to better simulate fluid and natural interactions with dynamic scenes with a common merit for which these techniques are popular in the implementations of real-time game applications.

These represent the mathematical basis for the simulation of the behavior of fluids, whereby the change in fluid velocity over time, under different forces, is described by NavierStokes equations. This equation forms the very necessary foundation for all the studies on fluid motion and finds its application in the Eulerian and Lagrangian methods of solution discussed above.

The two most important components of the Navier-Stokes equations are discussed below [5]. The first part is the momentum equation, describing how the velocity of a fluid changes under the action of forces: pressure gradients, viscosity (internal friction), and external forces like gravity. In other words, this equation describes how a fluid moves, responding to its internal properties and external forces such as wind or waves. This equation will give the developers of the game simulations a way to govern how a fluid reacts to characters or objects, or to terrain within a game setting; for example, this can make water splash when one jumps into the pool. The second part is the continuity equation, which furnishes the mass of the fluid that is conserved with the implication it can neither be created nor destroyed over time. This means in practice that the fluid flow is maintained constant and smooth. In game terms, that is to say, when water flows, there are no strange gaps or overlaps that make the simulation not realistic.

The fundamental equations for the calculation of the fluid motion, on the other hand, utilize the Navier-Stokes equations in the grid-based simulation of WCSPH and ISPH methods by computing particle displacements through the equation of momentum conservation and the continuity equation, with the latter providing density consistency over time. In turn, FVM applies these equations on a much smaller control volume so that the flow of fluids is accurately simulated in the case of large-scale simulations of bodies of water.

These complex equations, therefore, are simplified so that the developers can establish visually convincing simulations of phenomena without necessarily having to achieve other industries' required strict physical accuracy. According to [5], in most cases, speed and visual stability are more important than physical correctness in gaming.

3 Numerical Methods

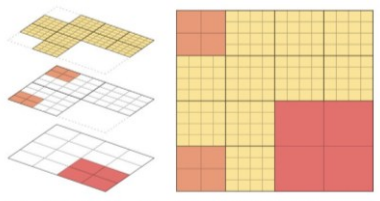
In the context of games, there are many numerical methods developed for fluid dynamics simulation to turn out properly this challenging trade-off between realism and performance. These methods try to emulate such phenomena in a way that their simulation would be visually realistic while keeping computational efficiency so as to be eligible in real-time game environments. In this section, some of the important numerical techniques in modern fluid simulations will be reviewed and summarized, each suited to a different aspect of fluid behavior: handling large bodies of water, dynamic fluid-object interactions, and ensuring real-time performance without giving up too much on visual fidelity.

3.1 DCGrid

In [1], "DCGrid: An Adaptive Grid Structure for Memory-Constrained Fluid Simulation on the GPU" introduces the Dynamic Constrained Grid (DCGrid), an advanced grid method for fluid simulation. This method enhances computational speed significantly while maintaining simulation accuracy.

Raateland et al. came up with a sparse grid structure ([1].Fig1), that has a hierarchical nature, and that structure is what the algorithm's data relies on mostly. In this system, the grid gets divided into levels, with each one having different resolutions, but those resolutions differ by about a factor of two between every level. The purpose behind this setup is to allocate memory and resources more freely while not affecting simulation precision. For example, where the fluid is smooth, a lower resolution is used; then, in areas with interaction or high turbulence, the resolution is raised for more accurate results. The grid is organized into blocks, with these blocks having sub-blocks, and inside them are multiple cells. But only the active ones, the cells that are in use, get memory. This design helps save storage and stops memory from being used unnecessarily.

There is a limit set on how many blocks are used, and all the levels have similar rules regarding memory usage. This prevents the algorithm from going over the available memory and ensures it operates smoothly within the constraints. The memory assignment happens in a linear way, and certain calculations are applied to map coordinates into memory positions through a hash table. This makes lookup times to remain at O(1), which is designed to keep time efficiency high. Another technique used by Raateland et al. for better performance is precomputed apron cells ([1].Fig2). Apron cells refer to cells around the block being processed. These are identified first in the fluid boundary task, which enables the direct use of their data in computations, and this leads to improved effectiveness.


(a) Hierarchy of sparse uniform grids. Colored cells are active.

(b) Flattened grid hierarchy into one dense multi-resolution grid, showing only active cells.

Fig. 1. Two-dimensional slice of the same hierarchy of sparsely populated uniform grids. The thicker lines indicate block boundaries.

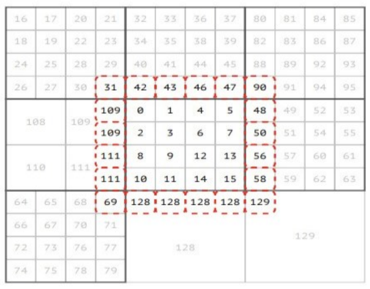
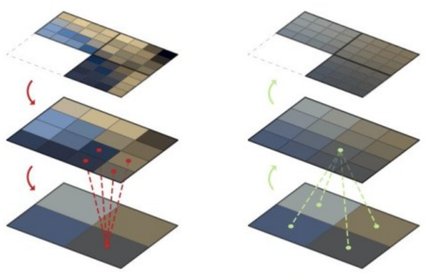


Fig. 2. Apron cell indices as calculated for the central block.

In addition to that, to keep the grid data consistent, they introduced operations of restriction and prolongation ([1].Fig3). Restriction collects data from blocks with higher resolutions, averaging them and sending the result to lower resolution blocks to keep the lower grid aligned with changes in the higher one. On the other hand, prolongation does the opposite by sending data from the lower resolution to the higher resolution blocks. This structure makes the algorithm good for large-scale parallel GPU tasks, keeping data in order while ensuring that calculations are fast.

Raateland et al. put forward a very comprehensive implementation for that algorithm. And DCGrid's topological adjustments rely on some key activities. Firstly, it is the priority score that plays a role here. They provided a kind of mathematical formula for it. This score defines how each grid cell resolution happens. The physical parameters, such as gradients in velocity and intensity of vorticity, are relevant for deciding. High-priority cells will have higher resolution, which ensures critical areas have more details. Then, topology adaptation takes place through both refinement as well as coarsening procedures. Refinement means new blocks are inserted, turning low-resolution grids into ones of higher resolution, and the prolongation part in this step helps the newly formed block have some reasonable initial state. Coarsening, on the other side, merges grids with high resolution into those of lower resolution. The efficiency of the hash table is kept by the algorithm refilling it periodically, which avoids inactive keys taking up the space and making search processes less efficient.


(a) Restriction operation updating cell values by averaging the values of their fine child cells.

(b) Prolongation of restricted values back to high-resolution grid.

Fig. 3. Restriction and prolongation operations performed after each other on the same data.

In the beginning, there is a global block limit being set up, and that controls the maximum number of blocks allowed at all levels. Then, how blocks are allocated in each level will depend on total memory availability and leftover space can be used for refining sub-blocks from lower levels that are more high-priority. The block re-arrangement ([1].Fig4) is another important thing in the algorithm's process. This happens after every timestep. The grid will be refined or coarsened based on those priority scores, where blocks with higher ones are refined, while lower ones get coarsened to save resources. To keep computational costs down, the algorithm has a move limit that tells how many adjustments happen each timestep, but this move limit isn't fixed; it will be adjusted according to system needs with a model that predicts needs simply. After each topological change, apron cells will be updated. That way, boundary cell data in numerical computations are kept right. At the same time, restriction and prolongation steps make sure data stays consistent between grids of high- and low-resolution levels.

Raateland et al. through a series of tests, looked at DCGrid's advantages. The tests had smoke and cloud simulations over some complicated areas. Various memory situations were tried in these experiments. The time for each frame to be calculated at 1080p was noted to be around 4 to 6 milliseconds, while at 4K the time increased, reaching about 10-15 milliseconds. Comparatively, algorithms like SPGrid or GVDB were slower, showing that DCGrid ran quicker and performed better when it came to GPU parallel computing.

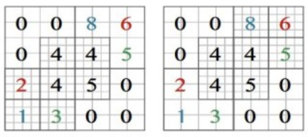


Fig. 4. Block re-arrangement. After each timestep, for each adjacent pair of grids in the hierarchy is considered. Fine blocks with low priority scores are matched with coarse subblocks with high priority scores. Then the fine blocks with low scores moved to the locations of the coarse subblocks with high scores.

DCGrid's main benefit is that it changes resolution as needed. It can adjust the grid size depending on different outside influences like collisions or similar conditions. Many optimizations for GPU computing have been added into the algorithm. Because of these factors, DCGrid not only gives correct simulations but also uses less memory and reduces the need for high computational power. This makes it stand out as one of the quicker fluid simulation methods around. For gaming, where simulating fluid in real time is important, DCGrid's capacity to handle large-scale parallel tasks makes it a useful and novel solution for doing fluid simulations in gaming environments.

3.2 Incompressible vs Weakly Compressible SPH

In [2], "Comparison of incompressible and weakly compressible SPH models for free-surface water flows" introduces Smoothed Particle Hydrodynamics (SPH), which is a novel numerical method developed to predict the behavior of liquids such as water. The two variants will be compared: Incompressible SPH (ISPH) versus Weakly Compressible SPH (WCSPH) that are applied, especially when the free surface is involved, such as waves and splashes. Unlike most of the grid-based methods, SPH models are fluid by particles capable of simulating complex and dynamic behaviors; this is very important in creating games when trying to replicate realistic water results.

This comparison between ISPH and WCSPH ascertains which method will be more efficient in the simulation of fluid in real-time environments. It states that ISPH maintains the volume of the fluid and is therefore accurate, whereas WCSPH allows small compressions, losing a bit of accuracy for gaining faster computation. Then, the paper goes ahead and compares both methods through dam-break scenarios and wave impact simulations that serve as benchmarks of performance in fluid simulation.

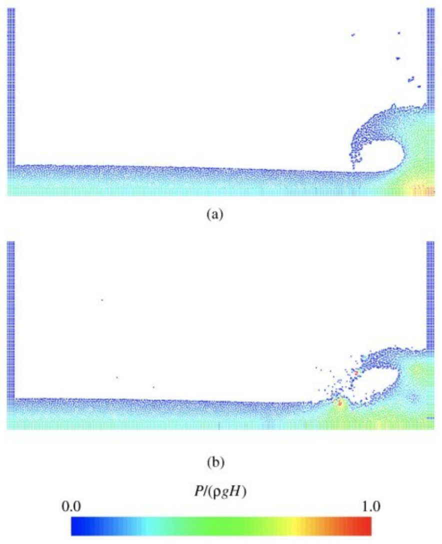


Fig. 5. $W = 2H$, $D = 5.366H$ dam-break, solution at $t^{p}g/H = 5.95$, for (a) WCSPH and (b) ISPH.

The dam-break scenarios in [2] were performed by considering WCSPH and ISPH methods under different conditions, such as the number of particles and boundary settings. Such a case study is always at the forefront of fluid simulation testing given the real-life situations of events that happen at a sudden collapse of a column of water. This displays an intricate behavior of wave formation and splash.

In the first dam-break test ([2].Fig5), the water column was released where basically similar results were obtained for both ISPH and WCSPH regarding the leading edge of the collapsing water and the way the water tumbled over obstacles. These comparisons indicated that both techniques performed fairly and in good agreement with the experimental data, reflecting the general trend of the fluid column. However, ISPH showed slower advance for the waterfront in some test cases; this is actually the additional computational cost that was paid by ISPH to satisfy incompressibility when the geometrical configuration is complicated. This makes ISPH computationally expensive since it is imperative to solve complex equations to conserve the volume of fluid.

In contrast, WCSPH yielded much smoother free-surface profiles without bearing a high load of computation. Among many great advantages which could be noted in the experiment was the fact that techniques like renormalization of particle densities every 20-time steps further enhanced smoothness and stability on the fluid surface within the simulation in WCSPH. This modification allowed WCSPH to maintain its performance while reducing its computational complexity. This, therefore, makes the process quite a bit more practical for such real-time applications as video games, where the speed of computation far outweighs minor inaccuracies.

The second dam-break test ([2].Fig6) simulated more difficult boundary and obstacle conditions, and the results again agreed with the first. In cases of pressure distribution and fluid flow around obstacles, ISPH continuously provided far more accurate simulations. However, WCSPH outperformed ISPH in speed, finishing the simulation a lot faster and maintaining smooth visual results. This further reinforces the suitability of WCSPH for real-time applications, such as in a game environment, by temporarily compromising the completeness of accuracy in fluids for visually appealing fluid interactions. Generally, it concluded that even though ISPH had better accuracy and stability, particularly with more detailed interactions-WCSPH can still be better applied in applications that require computational efficiency. Actually, this makes WCSPH much better for real-time fluid simulations in games, where one often needs to sustain smoothness and visual realism at the price of scientific precision.

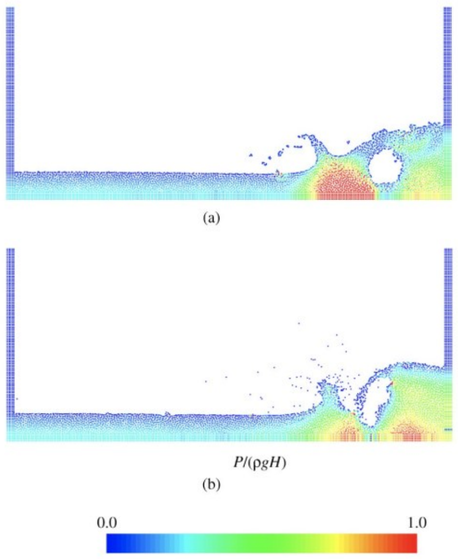


Fig. 6. $W = 2H$, $D = 5.366H$ dam-break, solution at $t^{P}g/H = 6.81$, for (a) WCSPH and (b) ISPH.

The wave impact simulations in [2] involved simulating waves against a vertical wall in order to see how ISPH and WCSPH handled pressure variations and free-surface interactions. This kind of test will be useful for game developers who work with coastal environments or dynamic water interactions in their scenes. In the setup, the models simulated waves acting against a solid barrier, consistent with real-world scenarios-that is, waves hitting a dock or rock.

The wave impacts have been generally well approximated by both ISPH and WCSPH. However, their performances diverged regarding pressure accuracy and smoothness of the free surface, as shown by ISPH, which presented a more detailed pressure distribution. Smooth and accurate pressure profiles were obtained along the wall by ISPH, as depicted in ([2].Fig7). This accuracy is important for applications in which it relies on highly accurate fluid behavior, such as high-end simulations or game cutscenes that have to have water interact with objects in great detail.

Contrarily, in WCSPH, it was possible to capture somewhat smoother free-surface results along the impact in cases with a high number of particles and dynamic movement of the water. On the other hand, WCSPH was remarked to show noisier pressure fields than ISPH, which turned out to be less reliable for the proper capturing of fine details in time, such as pressure fluctuations while the wave strikes the wall ([2].Fig8). With these minor inaccuracies, WCSPH could still develop a reasonably realistic overall wave impact behavior, thus being ready for real-time applications where the visual effect is often more important than strict accuracy.

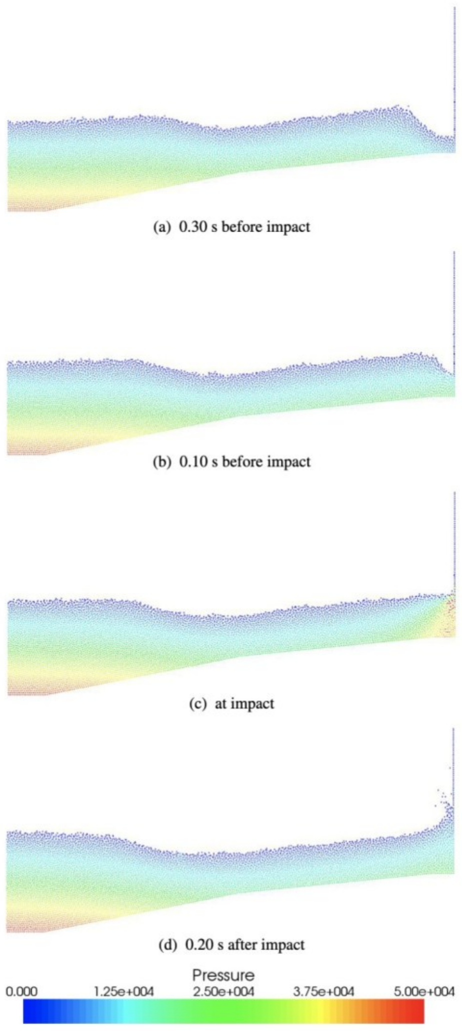


Fig. 7. Solution for 1.3m wave height ('flip-through' type imapct), computed using ISPH method

This trade-off between pressure accuracy in ISPH and free surface smoothness in WCSPH may indicate that WCSPH is more applicable to real-time game environments where the overall look and feel of the fluid play a greater role than minute pressure variations.

One possible conclusion that can be derived from [2] is that it has something to do with the trade-off between precision and speed through a comparative study of ISPH and WCSPH. Results by ISPH tend to be more accurate, particularly in pressure calculations and the rendering of realistic fluid movement in dam-break and wave impact tests. However, this comes at the cost of loss of computational efficiency, since ISPH needs to solve complex equations at every time step to maintain incompressibility in the fluids and it turns out to be hugely time-consuming. In cases where a high degree of accuracy is justified, such as engineering simulations or cinematic special effects, ISPH turns out very well.

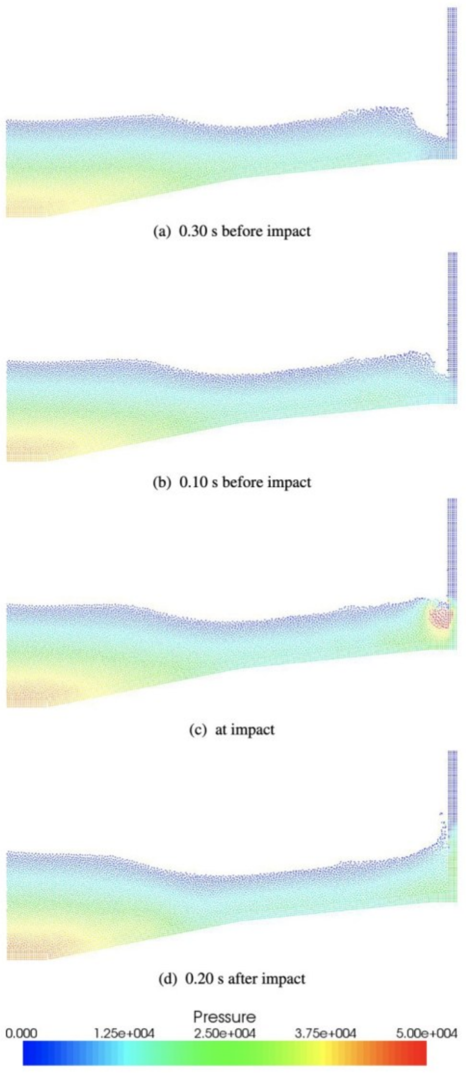


Fig. 8. Solution for 1.3m wave height ('flip-through' type impact), computed using WCSPH method

In contrast, WCSPH is much faster and quite efficient; thus, it serves better in real-time applications, as in the case of video games. Allowing for moderate compressibility, WCSPH reduces the computational load. Although it introduces minor inaccuracies, such as pressure oscillations and less smooth wave behavior, most of these often go unnoticed in dynamic situations. This trade-off becomes very important in gaming, where sustaining performance does not have to mean compromising on visuals.

In this context, [2] demonstrates that ISPH does not outperform WCSPH universally but is instead better suited for different contexts. Applications requiring higher accuracy should make use of ISPH, whereas WCSPH provides a good balance between realism and speed, hence being most appropriate for real-world fluid simulations, especially in the course of developing games.

Focusing on how WCSPH can be used in a real-world environment, [2] lays the very strong foundation required by individuals interested in applying particle-based fluid simulation in games. To developers with a need for dynamic water effects, WCSPH offers a great way to achieve visually plausible simulations without performance lag. The guidelines from [2] help guide decisions on when and how to use fluid simulations so that one can get the right balance between realism and speed.

While ISPH is more accurate, WCSPH's faster computation makes it way better for game development, since most aspects depend on real-time performance. It may be visualized from [2] that WCSPH allows the simulations of fluids with very minor visual sacrifices, hence being highly practical in wave and splash effects creation. Minor inaccuracies, like pressure fluctuations, usually stay imperceptible in fast-paced game environments, underlining WCSPH's suitability for real-time applications even further. In [2], this is Much of the detailed comparisons in [2] effectively illustrate why WCSPH is an ideal solution for fluid simulation in gaming.

3.3 Implicit Incompressible SPH and Its Improvements

IISPH [6] - This abbreviation refers to Implicit Incompressible Smoothed Particle Hydrodynamics, an important technique in the field of fluid simulation within computer-generated imagery. It is an extension of probably one of the most usable methods for the simulation of Lagrangian-based fluids, the SPH - Smoothed Particle Hydrodynamics. Because the Lagrangian viewpoint mainly follows the trajectory of a particle in time, this method is particularly suited for simulation runs of fluid scenarios where the boundaries are well defined, or where one wants to track the motion of individual particles with great precision, such as droplet collisions and liquid particle complex motions. IISPH has several reasons that make it outperform the standard SPH methods in simulating incompressible fluids; for one, the IISPH method uses the semi-implicit Euler method [6] to predict temporal changes of density. That is, the computation considers both current and next-step information; hence, it becomes more stable, allowing for larger time steps without affecting the accuracy of the simulation. Meanwhile, pressure is computed accurately by the IISPH method with the solution of pressure Poisson's equation in order to strictly maintain incompressibility.

In the IISPH approach, the boundary particles are treated as a different entity from the fluid particles. The early IISPH methods need to prepare special algorithms for generating the boundary particles and merge them into computation in a method different from the fluid particles. Obviously, designing separate computational processes and handling logic for these two types of particles enhanced computational complexity. Further, since these are two different computational processes, simulations such as solid liquefaction or liquid solidification make things worse.

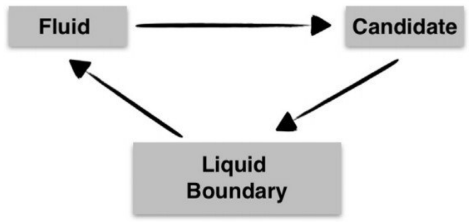


Fig. 9. Roles and role transitions that are considered in our implementation.

However, Cornelis et al. unified the representation of both boundary and fluid particles with the particle system. Since the new unified approach uses one process for all the particles, the solver implementation becomes simpler due to no distinctions between boundary or fluid particles. Furthermore, as both boundary and fluid particles can be represented by unified particles, it enables more natural animation of melting and solidification. This new solver can be used widely in games.

The work of Cornelis et al. proposed a new candidate particle in the support of fluid-to-liquid boundary transition. The main procedure of this technique comprises choosing the nearest fluid particle for each position of liquid boundary sample while updating it to a candidate particle. The latter then turns out to be a liquid boundary particle when proximal to the position.

The method proposed by Cornelis et al. supposed several roles attributed to particles. They categorize the particles into three types ([7].Fig9): fluid particles representing the fluid body, liquid boundary particles representing the boundary, and candidate particles that have to meet the constraints of the fluid density and be used while transitioning from fluid particles to liquid boundary particles. While generating a liquid boundary, for every sampled position on the rigid body, candidate particles are usually chosen as the nearest fluid particle. In practical applications, these three kinds can transform each other to meet the purpose of some effect such as boundary disruption ([2].Fig10).

In the experiments of paper [2], there are two main parameters β and α for improving the simulation effect. β is the animation parameter, which shows the velocity of the candidate particles animating to the specified positions on the liquid boundary. The constraint for that would be such that the maximum velocity of these should not be larger than those of the fluid particles to make the simulation stable. α changes the linear combination of the current particle velocity with the animation velocity for the particle velocity. By tuning α, the motion of the particle can be made more natural. Both these parameters will, in the end, strongly influence the performance and efficiency of the simulation.

Parameter β concerns the velocity increment of the candidate particles. However, setting the fluid velocities too high may violate the CFL condition and thus necessitates taking smaller time steps to maintain stability.

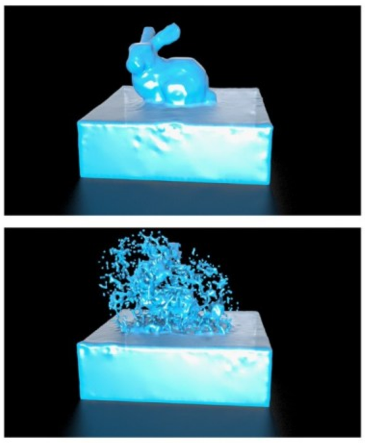


Fig. 10. Liquid boundary sampling. The first image shows the transition of a liquid boundary to fluid particles with an appropriate sampling using the proposed uniform grid. The second image shows the instabilities that occur in the transition of oversampled liquid boundaries to fluid particles.

For parameter α, the greater value represents completely overwriting the particle's velocity seen from the animation view. The smaller α reduces the animation velocity of the candidate particles; therefore, the animation duration increases. However, the larger α is not always better. Since the simulation works with incompressible fluids, the selected velocity field is divergence-free. This means that several iterations are necessary in order to remove the density errors caused by the parameter α. Such is the case of the experiments, where for $\alpha = 0.5$ only 18 iterations were necessary, while for $\alpha = 1.0$, the IISPH solver needed 62 iterations to resolve the density error. Therefore, $\alpha = 0.5$ was adopted in the paper to well balance performance and simulation stability. More precisely, a smaller value for α serves nothing in enhancing the simulation results; instead, it decelerates the assembling liquid boundary significantly.

Although the use of unified particles in IISPH through the work of Cornelis et al. has brought some benefits in performance or even the animation effects optimization when changing boundaries, some limitations still remain. It considers only rigid objects and additional considerations must be taken into account for deformable objects. The unified grid data structure also avoids density errors at liquid boundaries but introduces aliasing effects in surface reconstruction; postprocessing is necessary to reduce visual artifacts such as a lack of smoothness or realism. Finally, they currently assign only one fluid particle for each liquid boundary sample. Nevertheless, they do suggest that in certain conditions a single liquid boundary sample should be associated with multiple fluid particles in order to simulate fluid washing over a rigid object boundary.

3.4 Finite Volume Method

In [4], "A finite volume method parallelization for the simulation of free surface shallow water flows" introduces the Finite Volume Method (FVM), which is a numerical method for numerical solution of partial differential equations. It is widely used in fluid dynamics for simulating free surface shallow water flows. However, the computation of fluids dynamics requires a large memory size and a long computer code execution time, while it has depended on using serial computer environments for a long time [4]. Therefore, this article constructs a parallel algorithm using domain decomposition techniques, which is based on the very popular approximate Riemann solver of Roe, to improve the effectiveness of shallow water flows simulation.

The basic idea of the finite volume method is to divide the computational domain into a number of finite volume units (control bodies) and discretize them in the integral form of the conserved quantities in each control body. Meshing is the starting step of the finite volume method. The area is divided into multiple small control volumes (finite volumes), which can be of regular shape, such as rectangle or square, or irregular geometric shape. Through this process, researchers can transform complex continuous physical problems into discrete numerical problems. Subsequently, for each control volume, the partial differential equations are transformed into integral form by using the conservation laws of fluid dynamics. In this stage, integral operations are performed inside the control body based on conserved quantities such as mass, momentum or energy. Then, the volume integral of the control body is transformed into the area integral of the control body surface by using Gauss theorem. This transformation allows the conservation properties of the whole control body to be formulated in terms of the flow on its surface, so the solution of the problem translates into computing the flow in all directions on the surface of the control body. Conclusively, a linear system of equations is generated from the discrete equations of all control bodies, which are got from pervious steps. Researchers apply appropriate numerical methods to obtain the physical quantities inside the control volume. The data can be used to perform subsequent analysis and visualization.

Finite volume methods have several advantages over other numerical techniques. These methods, which combine the simplicity of finite differences with the geometrical flexibility of finite element methods, have received extensive attention due to their high performance in both subcritical and supercritical flow conditions [4]. As these methods rely on the integral form of conservation laws, it is simple to create numerical schemes that account for discontinuities. The primary challenge is thus to estimate the normal flows through each computational cell interface [4].

On a current serial platform, run-times for finite volume schemes in actual simulation can be very slow for precise results after the refinement of computational grid. Delis et al. adopt a common two-dimensional high-resolution explicit finite volume numerical approach to parallel platforms, utilizing developing programming paradigms. Explicit schemes often need fewer calculations per time step than implicit schemes, but the time saved on a pre-time-step basis may be wasted on a per-simulation period basis since the time step of explicit schemes is limited by the CFL stability requirement [4]. The parallel system uses domain decomposition and utilizes MPIstandard protocols for interprocessor communication.

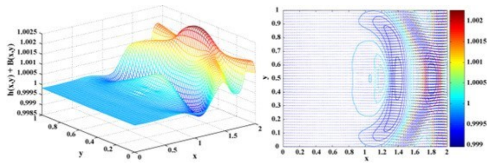


Fig. 11. Benchmark Problem 1: water depth (left) and contour plot with the velocity field (right).

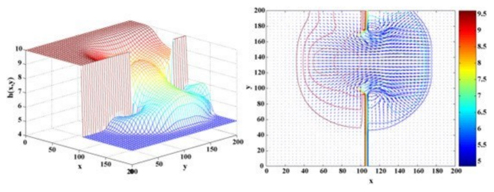


Fig. 12. Benchmark Problem 2: water depth (left) and contour plot with the velocity field (right).

To optimize performance, the workload should be equally distributed, and concurrency is maximized such that all processors are kept busy doing productive work while keeping communication overhead to a minimum. Delis et al. explore an implementation appropriate for distributed memory computers that divides the physical domain into sub-domains assigned to various processors. Two main characteristics of a distributed memory architecture are satisfied. Firstly, the method involves a few powerful processing nodes of the same type, connected by a high-speed network. Secondly, the partitioning of data and computation takes into account the current distributed memory structure while also keeping all processing nodes engaged during the calculation period.

The parallel algorithm is tested on three two-dimensional benchmark problems to evaluate the performance of fluid simulations. Scenario one is a wave propagation over topography, testing the well-balance property of numerical schemes([4].Fig11). Scenario two is a dam-break, showing that data communication has risen at each time step, due to the size of each sub-domain allotted to each processor, as well as the extended simulation duration([4].Fig12). Scenario three is a non-smooth bed topography([4].Fig13).

All the experiment results show that computation time is greatly reduced, and speedup approached linearity in most cases. The parallel strategy is the most portable solution since it works on any parallel architecture (in this case, two) and eliminates the need to partition the data into different files during the startup stage. However, deviations were noted due to cache memory issues and network connection types. One alternative approach is to use a parallel I/O method, which may result in improved performance, while the disadvantage is that in order to generate the appropriate files for the results' visualization, the data must be combined. In conclusion, the TVD scheme can perform well in fluid simulation, providing a good ratio between communication and computation as well. This capability can be utilized for more grid computing system implementations.

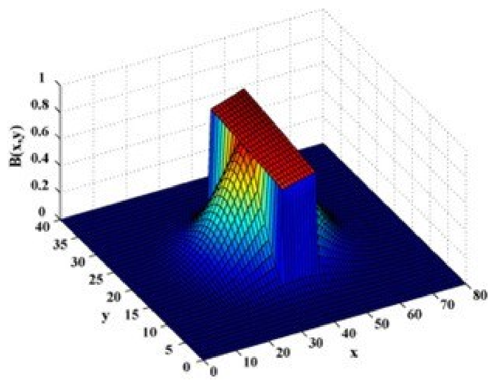


Fig. 13. Benchmark Problem 3: Non-smooth bed topography.

4 Conclusion

During recent years, the development of games bound fluid simulation as the main technology helpful in creating realistic and interactive virtual environments. This paper reviewed several methodologies for fluid simulation, where each of these methodologies has different strengths in different scenarios and thus R solutions to balance various computational performances and visual accuracies.

The DCGrid method, on the other hand, adjusts the grid resolution dynamically by enabling memory management and resource allocation, allowing it to have more computational focus on areas of intricate fluid interaction. This makes it ideal for real-time optimization of fluid performances. ISPH and WCSPH, by contrast, show particle-based methods that exhibit a trade-off between accuracy and speed. ISPH keeps the liquid incompressible, very accurate, and thus suitable for scenes that require precise control, such as cutscenes. In contrast, WCSPH allows small compressibility, thus improving computational efficiency, and hence better suited for fast-paced, real-time games where performance is key.

Furthermore, IISPH enhances ISPH with improved computational stability and larger time steps at no additional cost. Thus, it becomes more applicable to complex fluid interactions in dynamic scenes with high interactivity, especially in real-time games. Finally, the Finite Volume Method presents an efficient way to simulate large bodies of water by segmenting the fluid into control volumes. In particular, the FVM works where there are open-world games or scenes with vast water bodies that balance the level of fidelity and computational efficiency nicely.

This review of methodologies underlines how different techniques for the simulation of fluids can be put into practice depending on the exact needs of a game. Whether more accuracy or more performance is needed, the developer will have to make a choice depending on the needs of the game.

In the future, with increasingly higher demands for more immersive and graphically captivating game environments, fluid simulation will most probably improve. Newly developed hybrid methods that combine the strengths of both grid-based and particle-based techniques can provide superior solutions to the challenge of balancing accuracy and performance. Besides, by leveraging the advances in parallel computing on the GPU, more real-world optimizations can be achieved for real-time performance, allowing future games to create much more sophisticated and interactive fluids.

In the end, fluid simulation will remain at the core of the evolution of gaming, and mastering these techniques will be key to delivering the next generation of interactive and visually stunning gaming experiences.

Acknowledgment

We would like to extend our most sincere appreciation to Professor William Nace for his incessant guidance and support with regard to this literature review. His knowledge and professional feedback provided a great enhancement to our understanding of the field of fluid dynamics within video games. His mentorship and encouragement had a great bearing on both the structure and content of this review. Indeed, we are greatly indebted to his dedication and contribution, which greatly enhanced the quality and depth of our work.

References

[1] Wouter Raateland et al. "DCGrid: An Adaptive Grid Structure for Memory-Constrained Fluid Simulation on the GPU". In: *Proc. ACM Comput. Graph. Interact. Tech.* 5.1 (May 2022). DOI: 10. 1145/3522608. URL: https://doi.org/10.1145/3522608.

[2] Jason P. Hughes and David I. Graham. "Comparison of incompressible and weakly-compressible SPH models for free-surface water flows". In: *Journal of Hydraulic Research* 48.sup1 (2010), pp. 105–117. DOI: 10.1080/ 00221686.2010.9641251. eprint: https://doi.org/10.1080/ 00221686.2010. 9641251. URL: https://doi.org/10.1080/ 00221686.2010.9641251.

[3] Stefan Band et al. "Pressure Boundaries for Implicit Incompressible SPH". In: *ACM Trans. Graph.* 37.2 (Feb.2018). ISSN: 0730-0301. DOI: 10.1145/3180486. URL: https://doi.org/10.1145/ 3180486.

[4] Argiris I Delis and Emmanuel N Mathioudakis. "A finite volume method parallelization for the simulation of free surface shallow water flows". In: *Mathematics and Computers in Simulation* 79. 11 (2009), pp. 3339–3359.

[5] Jos Stam. "Real-time fluid dynamics for games". In: *Proceedings of the game developer conference*. Vol. 18. 11. 2003.

[6] Markus Ihmsen et al. "Implicit Incompressible SPH". In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 426–435. DOI: 10. 1109 / TVCG.2013.105.

[7] Jens Cornelis et al. "Liquid boundaries for implicit incompressible SPH". In: *Computers & Graphics* (Aug. 2015). DOI: 10.1016/j.cag.2015.07.022.