UDC 004.94 *http://dx.doi.org/10.30970/vam.2024.33.12376*

APPLICATION OF IISPH FOR INCOMPRESSIBLE FLUID DYNAMICS SIMULATION

O. Hrytsyshyn, V. Trushevskyy

Ivan Franko National University of Lviv,
1, Universytetska str., 79000, Lviv, Ukraine
e-mail: ostap.hrytsyshyn@lnu.edu.ua, valeriy.trushevsky@lnu.edu.ua

Smoothed Particle Hydrodynamics (SPH) is widely used in graphics and engineering for simulating fluids, viscous materials, and deformable solids. This paper presents the development and application of the Implicit Incompressible Smoothed Particle Hydrodynamics (IISPH) algorithm for simulating incompressible fluid dynamics. The IISPH method offers an efficient solution for enforcing incompressibility by solving the pressure Poisson equation implicitly, making it well-suited for large-scale simulations. The paper outlines the governing equations of Smoothed Particle Hydrodynamics (SPH), explains the key components of the IISPH solver, and demonstrates its effectiveness through the classical Taylor-Green vortex test case. The results show that the algorithm achieves real-time performance even with high particle counts, while maintaining accuracy and stability. Furthermore, a detailed analysis is provided to compare the numerical results with the analytical solution, highlighting the influence of particle resolution on simulation fidelity. The performance of the solver is validated across different particle counts, demonstrating its robustness for handling complex fluid dynamics simulations in engineering and computer graphics applications.

Key words: smoothed Particle Hydrodynamics, SPH, fluid simulation, Lagrangian methods, incompressibility, kernel functions, neighborhood search.

1. SPH FOUNDATION

Smoothed Particle Hydrodynamics (SPH) is a method used to discretize spatial field quantities and differential operators such as gradient and divergence. To facilitate this, the Dirac-δ function is introduced, which represents an idealized point mass. The Dirac-δ function is defined as:

$$\delta(r) = \begin{cases} \infty, & r = 0, \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

and satisfies the condition:

$$\int \delta(r)dv = 1.$$

This function allows the convolution of a continuous compactly supported function $A(\mathbf{x})$ with the Dirac-δ function, resulting in:

$$A(\mathbf{x}) = (A * \delta)(\mathbf{x}) = \int A(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')d\mathbf{x}'. \tag{2}$$

This forms the basis for the discretization in SPH, originally proposed by Gingold & Monaghan (1977) [1].

2. Continuous approximation

To approximate the integral in (2) for numerical computation, we replace the Dirac-δ function with a smoothing kernel $W(r,h)$. This kernel is used to approximate the integral as follows:

$$A(\mathbf{x}) \approx (A * W)(\mathbf{x}) = \int A(\mathbf{x}')W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}', \tag{3}$$

where $W(r,h)$ satisfies several conditions:

- Normalization condition:

$$\int_{\mathbb{R}^d} W(r,h)\, d\mathbf{r}' = 1$$

- Dirac-δ condition:

$$\lim_{h\to 0} W(r,h) = \delta(r)$$

- Positivity condition:

$$W(r,h) \geq 0$$

- Symmetry condition:

$$W(r,h) = W(-r,h)$$

- Compact support condition:

$$W(r,h) = 0 \quad \text{for } \|r\| \geq \hat{h}.$$

To refine the approximation of the integral with the smoothing kernel W, we expand $A(\mathbf{x}')$ around $\mathbf{x}$. This leads to:

$$\begin{aligned}(A * W)(\mathbf{x}) = &\int \left[A(\mathbf{x}) + \nabla A|_{\mathbf{x}} \cdot (\mathbf{x}'-\mathbf{x})\right] W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}' \\ &+ \mathcal{O}((\mathbf{x}'-\mathbf{x})^2).\end{aligned} \tag{4}$$

This simplifies to:

$$A(\mathbf{x}) \int W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}' + \nabla A|_{\mathbf{x}} \int (\mathbf{x}'-\mathbf{x})W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}'. \tag{5}$$

The approximation is accurate to first-order if the first term integrates to 1 (normalization condition) and the second term vanishes, which is ensured by the symmetry of the kernel. Different types of smoothing kernels and their properties can be found in [4].

3. Discretization

In SPH, the continuous integral from (2) is approximated by a discrete sum over particle samples, as shown in (6). This results in the discrete approximation:

$$\langle A(\mathbf{x}_i)\rangle \approx \sum_j A_j \frac{m_j}{\rho_j} W_{ij}, \tag{6}$$

where W_{ij} represents the kernel function evaluated at the positions of particles i and j. The first-order accuracy of this discretization depends on satisfying the conditions:

$$\sum_j \frac{m_j}{\rho_j} W_{ij} = 1 \quad \text{and} \quad \sum_j \frac{m_j}{\rho_j} (\mathbf{x}_j - \mathbf{x}_i) W_{ij} = 0.$$

While these conditions may not always hold due to the particle sampling pattern, normalization techniques can restore 0th- or 1st-order consistency for more accurate simulations [3].

3.1. Discretization of differential operators

In SPH, the gradient of a field can be approximated by discretizing the differential operators. The gradient of the field is given by:

$$\nabla A_i \approx \sum_j A_j \frac{m_j}{\rho_j} \nabla W_{ij}. \tag{7}$$

For higher-dimensional functions, other operators like divergence and curl can be discretized similarly:

$$\nabla \cdot A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \cdot \nabla W_{ij}, \tag{8}$$

$$\nabla \times A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \times \nabla W_{ij}. \tag{9}$$

However, direct computation using these formulas often results in poor accuracy. The difference formula, defined as:

$$\nabla A_i \approx \sum_j \frac{m_j}{\rho_j} (A_j - A_i) \nabla W_{ij}, \tag{10}$$

provides a better approximation and can be improved by solving a small linear system:

$$\nabla A_i \approx L_i \left(\sum_j \frac{m_j}{\rho_j} (A_j - A_i) \nabla W_{ij} \right), \tag{11}$$

where L_i is computed from the inverse of:

$$L_i = \left(\sum_j \frac{m_j}{\rho_j} \nabla W_{ij} \times (\mathbf{x}_j - \mathbf{x}_i) \right)^{-1}. \tag{12}$$

Another approach is the symmetric formula [3], which approximates the gradient as:

$$\nabla A_i \approx \rho_i \sum_j m_j \left(\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2} \right) \nabla W_{ij}. \tag{13}$$

While this formula does not exactly reproduce constant or linear gradients, it is advantageous for simulations as it conserves linear and angular momentum, making it

more robust for physical simulations. The error in the symmetric gradient is governed by how much

$$\sum_j m_j \left(\frac{1}{\rho_i^2} + \frac{1}{\rho_j^2} \right) \nabla W_{ij} \approx 0 \tag{14}$$

deviates from zero. This formula tends to reorder particles until the condition is satisfied. In summary, the difference formula yields more accurate gradients, while the symmetric formula is more stable due to its conservation of momentum.

4. Governing equations

This section outlines the fundamental equations governing fluid dynamics, including the continuity equation, which ensures mass conservation; the conservation law of linear momentum, extending Newton's second law to continuous media; and the Navier-Stokes equation, which models viscous fluid motion.

4.1. Continuity equation

The continuity equation governs the change in mass density, ρ, of an object over time. It is given by:

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot \mathbf{v}), \tag{15}$$

where $\frac{D}{Dt}$ represents the material derivative. This equation plays a crucial role in simulations involving incompressible materials. For such materials, the condition simplifies to:

$$\frac{D\rho}{Dt} = 0 \quad \Leftrightarrow \quad \nabla \cdot \mathbf{v} = 0. \tag{16}$$

This constraint ensures that the mass density remains constant and must be maintained at all times and at every point within the material. The continuity equation and its use in Smoothed Particle Hydrodynamics (SPH) for fluid simulations are thoroughly discussed by Monaghan [2].

4.2. Conservation law of linear momentum

The conservation of linear momentum can be viewed as an extension of Newton's second law for continuous media, often referred to as the equation of motion. This principle states that the rate of change of momentum of a material particle is equal to the sum of all internal and external volume forces acting on it, expressed as:

$$\rho \frac{D^2\mathbf{x}}{Dt^2} = \nabla \cdot \mathbf{T} + \mathbf{f}_{\text{ext}}, \tag{17}$$

where $\mathbf{T}$ is the stress tensor and $\mathbf{f}_{\text{ext}}$ represents body forces (forces per unit volume). This relation is independent of the material composition since the material behavior is captured in the stress tensor, which is defined by constitutive laws.

4.3. Navier-Stokes equation

For incompressible fluid flow, a common constitutive law is given by:

$$\mathbf{T} = -p\mathbf{I} + \mu(\nabla\mathbf{v} + \nabla\mathbf{v}^T), \tag{18}$$

where p is the pressure, μ is the dynamic viscosity, and $\mathbf{v}$ is the velocity field. In the case of strict incompressibility, p can act as a Lagrange multiplier, ensuring the condition from Eq. (16). If incompressibility is not strictly enforced, a state equation can be introduced to relate pressure to density changes, for example:

$$p = p(\rho). \tag{19}$$

One commonly used state equation is derived from the ideal gas law, expressed as:

$$p(\rho) = k\left(\frac{\rho}{\rho_0} - 1\right), \tag{20}$$

where k is a stiffness constant and ρ_0 is the reference density. Substituting the constitutive law (Eq. (18) into the equation of motion (Eq. (17) gives the incompressible Navier-Stokes equation:

$$\rho\frac{D\mathbf{v}}{Dt} = -\nabla p + \mu\nabla^2\mathbf{v} + \mathbf{f}_{\text{ext}}. \tag{21}$$

The Navier-Stokes equation, which describes the motion of viscous fluid substances, is a fundamental equation in fluid dynamics. Its application in the context of Smoothed Particle Hydrodynamics (SPH) for incompressible flow is well discussed in foundational works such as Monaghan [2] and Morris [7].

5. Discretization with implicit incompressible SPH (IISPH)

The Implicit Incompressible SPH (IISPH) algorithm, as proposed in [6], is an alternative discretization method for simulating incompressible flows. The method solves the pressure implicitly to ensure incompressibility, making it efficient for particle-based simulations.

The IISPH method is based on solving the pressure Poisson equation (PPE), reformulated for each particle i:

$$\Delta t^2\nabla^2 p_i = \rho^0 - \rho_i^*. \tag{22}$$

The source term, ρ_i^*, for SPH is computed as:

$$\rho^0 - \rho_i^* = \rho^0 - \rho_i - \Delta t\sum_j m_j(\mathbf{v}_i^* - \mathbf{v}_j^*)\cdot\nabla W_{ij}, \tag{23}$$

where $\mathbf{v}_i^* = \mathbf{v}_i + \Delta t\mathbf{a}_i^{\text{nonp}}$, and the pressure acceleration is:

$$\mathbf{a}_i^p = -\frac{1}{\rho_i}\nabla p_i = -\sum_j m_j\left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij}. \tag{24}$$

The IISPH discretization of the PPE becomes:

$$\Delta t^2 \sum_j m_j \left(\mathbf{a}_i^p - \mathbf{a}_j^p\right) \cdot \nabla W_{ij} = \rho^0 - \rho_i^*. \tag{25}$$

The terms used in the pressure update are defined as follows:

$$(Ap)_i = \Delta t^2 \sum_j m_j \left(a_i^p - a_j^p\right) \cdot \nabla W_{ij}, \tag{26}$$

where a_i^p is the pressure acceleration at particle i, and ∇W_{ij} is the gradient of the kernel function between particles i and j.

The source term s_i is given by:

$$s_i = \rho^0 - \rho_i^*, \tag{27}$$

where ρ^0 is the rest density, and ρ_i^* is the predicted density at particle i.

This system is solved iteratively for pressure, where pressure updates ensure that the divergence of the pressure acceleration leads to the correction of density changes. The final update for pressure is computed using a Jacobi solver:

$$p_i^{(l+1)} = (1-\omega)\, p_i^{(l)} + \frac{\omega}{a_{ii}} \left(s_i - \sum_{j \neq i} a_{ij} p_j^{(l)} \right), \tag{28}$$

where ω is the relaxation coefficient, typically set to 0.5, and a_{ii} and a_{ij} are matrix elements computed from the pressure accelerations.

The update in Eq. (28) requires the diagonal element a_{ii}, which can be calculated by accumulating all coefficients of p_i. The diagonal element is:

$$a_{ii} = -\Delta t^2 \sum_j m_j \left(\sum_j \frac{m_j}{\rho_j^2} \nabla W_{ij} \right) \cdot \nabla W_{ij} - \Delta t^2 \sum_j m_j \left(\frac{m_i}{\rho_i^2} \nabla W_{ij} \right) \cdot \nabla W_{ij}. \tag{29}$$

5.1. Stop criterion

There is no widely accepted method for determining when to stop the Jacobi iterations. The iterations can either be stopped after a fixed number of steps or when the predicted density error falls below a certain threshold. The predicted relative density error for particle i, based on the pressure field at iteration l, is calculated as:

$$\rho_i^{\text{err},*} = \frac{(Ap)_i - s_i}{\rho^0} = \frac{(Ap)_i + \rho_i^* - \rho^0}{\rho^0}. \tag{30}$$

Typically, the stopping criterion is based on the average of all relative density errors, $\rho_i^{\text{avg,err},*}$, which is computed as:

$$\rho_i^{\text{avg,err},*} = \frac{1}{n} \sum_i \left| \rho_i^{\text{err},*} \right|. \tag{31}$$

As proposed in [6], a common threshold is to stop the iterations when $\rho_i^{\text{avg,err},*}$ is less than 0.1%, ensuring the overall fluid volume fluctuation is below this value.

6. Implementation of IISPH solver

The IISPH solver iterates over particles to calculate pressure corrections and ensure incompressibility. The source term s_i and the diagonal element a_{ii} are computed once at the start. During each iteration l, the pressure Laplacian $(Ap)_i^{(l)}$ is computed in two stages.

Algorithm 1 Pressure Computation with the IISPH Solver

for all particles i **do**
  Compute diagonal element a_{ii} with Eq. (29)
  Compute source term s_i with Eq. (27)
  Initialize pressure $p_i^{(0)} = 0$
end for
Set $l = 0$
repeat
  for all particles i **do**
    Compute pressure acceleration $a_i^{p(l)}$ with Eq. (24)
  end for
  for all particles i **do**
    Compute Laplacian $(Ap)_i^{(l)}$ with Eq. (26)
    Update pressure $p_i^{(l+1)}$ with Eq. (28)
  end for
  Increment $l = l + 1$
until $\rho_i^{\text{avg, err},*} < 0.1\%$

7. Neighborhood search

In particle-based simulations, evaluating the force terms for all particles can be inefficient, leading to a computational complexity of $O(n^2)$, where n is the number of particles. To improve efficiency, smoothing kernels with compact support are employed, where interactions occur only between neighboring particles within a specific radius. This reduces the complexity to $O(nm)$, where m is the number of neighboring particles.

There are algorithms designed to solve this problem more efficiently, such as the one using compact hashing described by [5]. The idea is to place a uniform grid over the domain with a grid cell size equal to the kernel support radius, reducing the complexity to $O(n)$ by efficiently querying only neighboring cells. On a CPU, this approach is highly efficient, but adapting it for GPU architectures requires further optimizations.

7.1. Compact hashing approach for neighborhood search

The method described in [5] introduces an efficient neighborhood search approach for Smoothed Particle Hydrodynamics (SPH) simulations on multi-core CPUs. The authors propose an optimized spatial hashing scheme, referred to as *compact hashing*, to address the inefficiencies of traditional uniform grids and spatial hashing methods.

7.1.1. Overview of compact hashing

In this method, a uniform grid is constructed over the simulation domain, where the cell size equals the kernel support radius h. Each particle is assigned to a grid cell based on its spatial coordinates. A key improvement is the use of a compact list to store only non-empty cells. Instead of allocating memory for all possible cells in the domain, the compact hashing technique dynamically allocates memory only for cells that contain particles. Each cell in the hash table points to a compact list entry, reducing memory overhead and improving efficiency for sparsely populated domains.

The structure of the data and its mapping is illustrated in Figure 1. The first array represents the hash table, which stores handles pointing to the second array, a compact list of non-empty cells. Each non-empty cell further points to an array of particle indices for particles assigned to that cell. Memory is dynamically allocated for the particle indices in each non-empty cell, with a fixed allocation size k for each cell.

The hash function used to map 3D spatial coordinates (x, y, z) to a 1D hash table index is:

$$c = \left(\left\lfloor \frac{x}{d} \right\rfloor p_1 \oplus \left\lfloor \frac{y}{d} \right\rfloor p_2 \oplus \left\lfloor \frac{z}{d} \right\rfloor p_3 \right) \bmod m,$$

where d is the cell size, p_1, p_2, p_3 are large prime numbers, and m is the hash table size. This function minimizes collisions by spreading the data evenly across the table. While this description is for 3D space, the same approach applies to 2D simulations by omitting the z-coordinate.

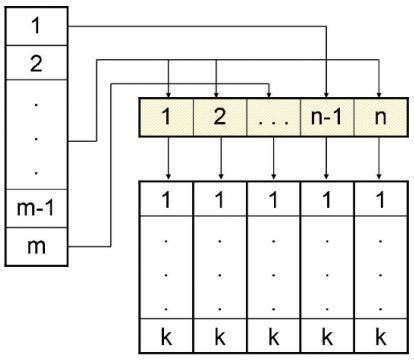


Fig. 1. Compact hashing. The hash table (size m) stores handles pointing to a compact list of n non-empty cells (yellow). Each cell reserves memory for k entries, with total memory usage $O(n \cdot k + m)$. The neighborhood query only processes these n non-empty cells. Figure taken from [5]

7.1.2. Neighborhood query

To find neighbors for a particle, the algorithm evaluates only the 27 (9 for 2D space) cells in the immediate vicinity of the particle's grid cell. This ensures that the computational cost of the query scales linearly with the number of particles, i.e., $O(n)$.

7.1.3. Limitations and applicability

Although compact hashing is highly efficient on CPUs, the hash function does not preserve spatial locality, which can lead to increased memory transfers. The method performs best when combined with techniques such as reordering particles to enforce spatial locality.

7.2. Applying compact hashing to GPUs

Adapting compact hashing for GPUs requires leveraging compute shaders for efficient parallelism while addressing GPU-specific challenges such as memory management, hash collisions, and sorting.

To ensure predictable memory access and reduce runtime overhead, the hash table is pre-allocated with a size equal to the number of particles. While this approach does not eliminate hash collisions, it minimizes memory allocation overhead and allows for more efficient parallel processing compared to dynamic memory allocation.

The GPU implementation processes particles in parallel, with grid cell indices calculated for all particles simultaneously. Neighbor searches are also performed in parallel, querying the surrounding cells to identify interactions.

A crucial step in this approach is sorting particles based on their grid indices to improve memory access patterns during neighbor searches. Sorting is a challenging part of the implementation, especially on GPUs, where performance depends on efficient parallel algorithms. To address this, it is proposed to use Bitonic sort [9], a parallel sorting method optimized for GPUs.

7.3. Bitonic sort for efficient GPU sorting

Sorting particles by their grid indices is an important step in the GPU implementation of compact hashing. It ensures that particles in the same or neighboring grid cells are grouped together, which makes neighbor searches much faster. Bitonic sort, introduced by Batcher [9], is a parallel sorting algorithm that works well on GPUs because it is highly structured and easy to parallelize.

A sequence is called *bitonic* if it first increases and then decreases, or vice versa. Bitonic sort uses this property to sort the sequence into order. The algorithm works by repeatedly comparing pairs of elements and swapping them to divide the sequence into smaller sorted parts. These smaller parts are then combined into a fully sorted sequence.

On GPUs, Bitonic sort is implemented using compute shaders, where each thread handles a specific comparison and swap. This allows many operations to run at the same time, making the sorting process much faster than on a CPU.

8. Results

This section presents the validation and performance of the Smoothed Particle Hydrodynamics (SPH) method applied to the Taylor-Green vortex problem, a benchmark for incompressible Navier-Stokes simulations. The results include an overview of the vortex's initial conditions, its analytical solution, and the evolution of the flow over time, highlighting the decay due to viscosity. Simulations demonstrate the method's ability to accurately capture vortex dynamics and its suitability for real-time fluid simulations.

Algorithm 2 Bitonic Sort

```
function BitonicSort(array, up)
    Divide array into two halves
    Sort first half with BitonicSort(first_half, True)
    Sort second half with BitonicSort(second_half, False)
    return BitonicMerge(first_half + second_half, up)
end function
function BitonicMerge(array, up)
    CompareAndSwap(array, up)
    Merge first half with BitonicMerge(first_half, up)
    Merge second half with BitonicMerge(second_half, up)
    return first_half + second_half
end function
function CompareAndSwap(array, up)
    for all indices i in the first half of array do
        if (array[i] > array[i + n/2]) == up then
            Swap array[i] and array[i + n/2]
        end if
    end for
end function
```

8.1. Taylor-Green vortex

The Taylor-Green vortex is a classical test case for validating numerical methods applied to the incompressible Navier-Stokes equations. This vortex flow is characterized by periodic vortex structures that decay over time due to viscosity. It is commonly used to benchmark the accuracy and stability of fluid simulation methods, such as Smoothed Particle Hydrodynamics (SPH). The test case was originally introduced by Taylor and Green (1937), making it a widely used benchmark in computational fluid dynamics [8].

8.1.1. Initial conditions

The initial velocity field for the 2D Taylor-Green vortex in a domain $[0, 2\pi] \times [0, 2\pi]$ is defined as:

$$u(x, y, 0) = \sin(x)\cos(y), \tag{32}$$

$$v(x, y, 0) = -\cos(x)\sin(y), \tag{33}$$

where u and v are the velocity components in the x and y directions, respectively. The initial pressure is assumed to be constant, and the initial density is typically set to a uniform value.

8.1.2. Analytical solution

As time evolves, the velocity field decays due to the action of viscosity. The exact solution for the velocity components at any time t is given by:

$$u(x, y, t) = \sin(x)\cos(y)e^{-2\nu t}, \tag{34}$$

$$v(x, y, t) = -\cos(x)\sin(y)e^{-2\nu t}. \tag{35}$$

8.2. Taylor-Green vortex simulation

This subsection presents the evolution of the Taylor-Green vortex using 6,000 particles in a $[0, 2\pi] \times [0, 2\pi]$ domain. The Figure 2 show the simulation results at different time frames from $t = 0$ to $t = 5$, where the vortex fully decays by $t = 5$ due to the effect of viscosity.

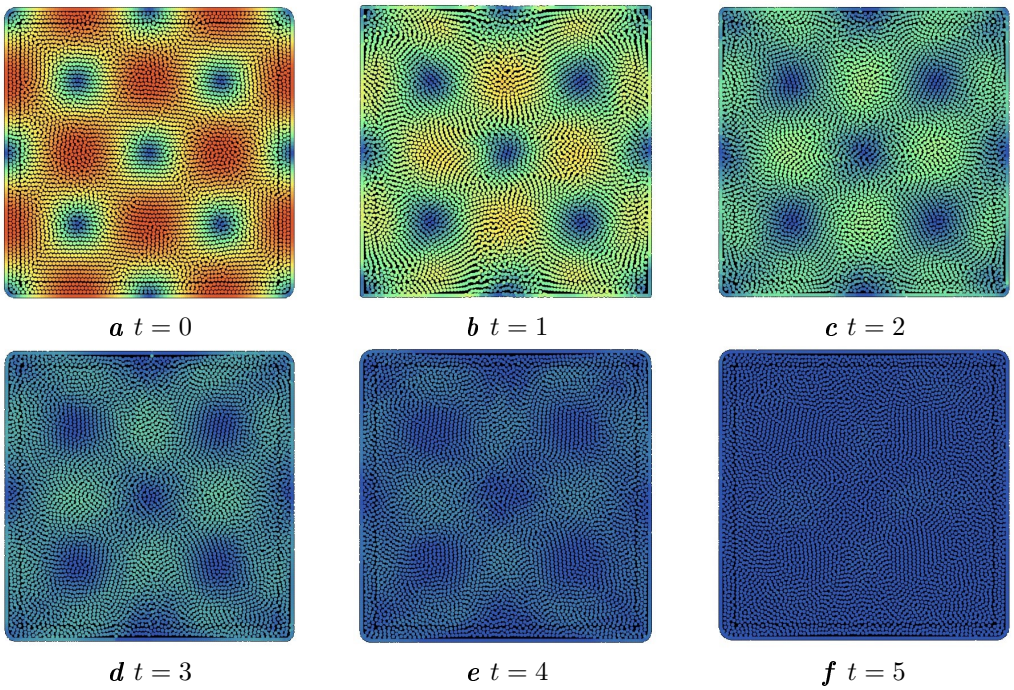


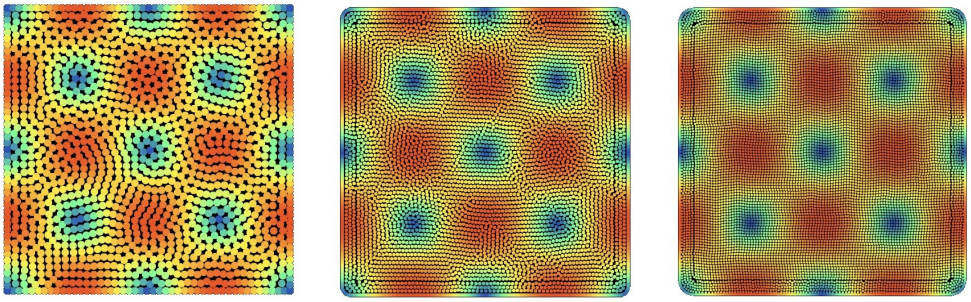
Fig. 2. Taylor-Green vortex simulation at different times, showing the progression of the vortex decay from $t = 0$ to $t = 5$

8.3. Comparison to analytic solution

In this section, we compare the simulation results for the Taylor-Green vortex problem to the analytic solution at different particle resolutions: 1500, 6000, and 12000 particles. The objective is to assess how increasing the number of particles affects the accuracy of the simulation and the level of detail captured in the flow field.

8.4. Initial state comparison at $t = 0$

Figure 3 presents the velocity field of the Taylor-Green vortex at the initial time $t = 0$ for different particle counts: 1500, 6000, and 12000 particles. As the number of particles increases, the simulation captures finer details of the flow structure, illustrating the benefit of higher resolution in accurately representing the vortex dynamics.



1500 particles 6000 particles 12000 particles

Fig. 3. Velocity field comparison at $t = 0$ for different particle resolutions. As the number of particles increases, more details of the flow are captured

8.5. Error analysis over time

The accuracy of the simulation can be quantified by comparing the simulated velocity field to the analytic solution using the root mean square error (RMSE). Figure 4 shows the RMSE evolution over time for the three different particle resolutions: 1500, 6000, and 12000. As expected, higher particle counts result in lower RMSE, indicating better accuracy. The results demonstrate that simulations with more particles capture the decay of the vortex more precisely, especially in the later stages of the simulation.

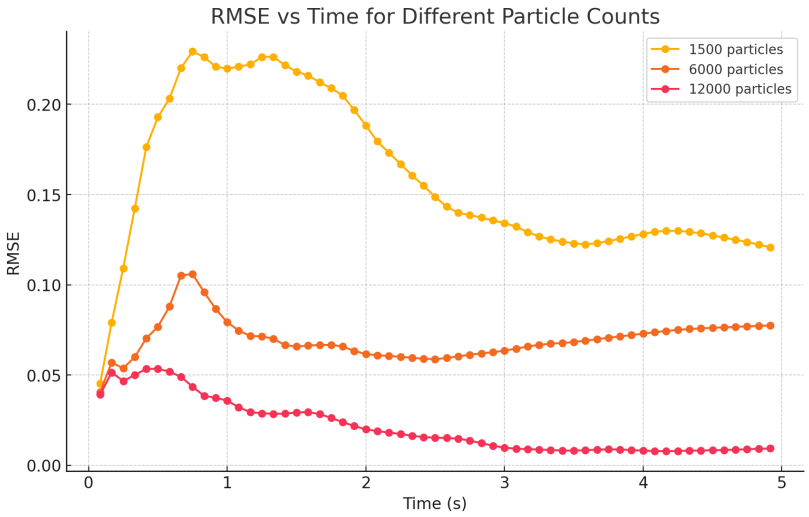


Fig. 4. RMSE comparison between the simulated and analytic velocities for different particle resolutions over time. The accuracy improves as the number of particles increases

As seen in Figure 4, the 12000-particle simulation closely follows the analytic solution, with minimal error throughout the time evolution. In contrast, the 1500-particle simulation exhibits higher RMSE, particularly in the later stages, where the resolution is insufficient to accurately model the vortex decay.

All simulation steps are performed with a fixed time step, and the computational time for each step remains approximately 0.15 ms, regardless of particle count, whether the simulation uses 1500 or 12000 particles. In order to enhance accuracy and reduce numerical error, especially in capturing fine details of fluid dynamics, we execute five physics simulation steps for each rendered frame. This ensures that smaller time steps are utilized, leading to a more precise evolution of the velocity field over time. By performing multiple simulation steps per frame, the method reduces the cumulative error, particularly in the later stages of the vortex decay, where finer time resolution plays a critical role in maintaining simulation fidelity. As a result, even high-resolution simulations maintain real-time performance while achieving greater accuracy.

9. Conclusion

This paper presents the application of the Implicit Incompressible Smoothed Particle Hydrodynamics (IISPH) algorithm to simulate incompressible fluid dynamics. By utilizing the IISPH approach, we ensured accurate enforcement of incompressibility while maintaining computational efficiency. The Taylor-Green vortex problem was used as a benchmark to validate the solver's accuracy and performance across different particle resolutions.

The results demonstrated that as the number of particles increased from 1500 to 12000, the simulation's accuracy improved, with finer details of the vortex structure being captured. The root mean square error (RMSE) analysis confirmed that higher particle counts better matched the analytic solution, especially during the later stages when vortex decay becomes more prominent. Additionally, even at the highest resolution with 12000 particles, the simulation ran in real-time, demonstrating the efficiency of the IISPH solver in handling large particle counts without sacrificing performance.

To further improve computational efficiency, GPU-based techniques were incorporated into the implementation. Compact hashing was adapted for GPUs to accelerate neighborhood searches, and Bitonic sort was used to efficiently sort particles by their grid indices. These optimizations allowed the solver to maintain real-time performance even at higher resolutions, making it scalable for large-scale simulations.

Future work could explore extending the IISPH method to simulate multi-phase fluids, handling more complex boundary conditions, and applying the solver to three-dimensional flows. Additionally, integrating adaptive particle refinement techniques could improve efficiency while maintaining high accuracy in regions of interest.

Overall, the IISPH method, enhanced with GPU optimizations such as compact hashing and Bitonic sort, proves to be a robust and effective tool for simulating incompressible fluids in real-time, making it suitable for a wide range of applications in both engineering and computer graphics.

References

1. Gingold R.A. Smoothed particle hydrodynamics: theory and application to non-spherical stars / R.A. Gingold, J.J. Monaghan // Monthly Notices of the Royal Astronomical Society. – 1977. – Vol. 181, № 3. – P. 375–389.
2. Monaghan J.J. Smoothed particle hydrodynamics / J.J. Monaghan // Annual Review of Astronomy and Astrophysics. – 1992. – Vol. 30. – P. 543–574.
3. Price D.J. Smoothed particle hydrodynamics and magneto-hydrodynamics / D.J. Price // Journal of Computational Physics. – 2012. – Vol. 231, № 3. – P. 759–794.

4. Liu M. Smoothed particle hydrodynamics (SPH): an overview and recent developments / M. Liu, G. Liu // Archives of Computational Methods in Engineering. – 2010. – Vol. 17, № 1. – P. 25–76.
5. Ihmsen M. A parallel SPH implementation on multi-core CPUs / M. Ihmsen, N. Akinci, M. Becker, M. Teschner // Computer Graphics Forum. – 2011. – Vol. -30, № 1. – P. 99–112.
6. Ihmsen M. Implicit incompressible SPH / M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, M. Teschner // IEEE Transactions on Visualization and Computer Graphics. – 2014. – Vol. 20, № 3. – P. 426–435.
7. Morris J.P. Modeling low Reynolds number incompressible flows using SPH / J.P. Morris, P.J. Fox, Y. Zhu // Journal of Computational Physics. – 1997. – Vol. 136, № 1. – P. 214–226.
8. Taylor G.I. Mechanism of the production of small eddies from large ones / G.I. Taylor, A. E. Green // Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences. – 1937. – Vol. 158, № 895. – P. 499–521.
9. Batcher K.E. Sorting networks and their applications / K.E. Batcher // Proceedings of the AFIPS Spring Joint Computer Conference. – 1968. – Vol. 32. – P. 307–314.

Article: received 07.10.2024
revised 24.10.2024
printing adoption 14.11.2024

ЗАСТОСУВАННЯ IISPH ДЛЯ СИМУЛЯЦІЇ ДИНАМІКИ НЕСТИСЛИВОЇ РІДИНИ

О. Грицишин, В. Трушевський

Львівський національний університет імені Івана Франка,
вул. Університетська 1, Львів, 79000, Україна
e-mail: ostap.hrytsyshyn@lnu.edu.ua, valeriy.trushevsky@lnu.edu.ua

Метод гідродинаміки згладжених частинок (SPH) широко використовують в графіці та інженерії для симуляції рідин, в'язких матеріалів і деформованих тіл. Подано розробку та застосування алгоритму неявної нестисливої гідродинаміки згладжених частинок (IISPH) для симуляції динаміки нестисливих рідин. Метод IISPH забезпечує ефективне вирішення проблеми підтримання нестисливості шляхом неявного розв'язку рівняння Пуассона для тиску, що робить його придатним для великомасштабних симуляцій. Описано основні рівняння методу SPH, пояснено ключові компоненти розв'язувача IISPH і продемонстровано його ефективність на прикладі класичної задачі вихору Тейлора-Гріна. Результати підтверджують, що алгоритм досягає ефективності в реальному часі навіть за великої кількості частинок, зберігаючи точність і стабільність. Крім того, надано детальний аналіз для порівняння чисельних результатів з аналітичним розв'язком, що наголошує на впливі роздільної здатності частинок на точність симуляції. Ефективність алгоритму підтверджена на різних кількостях частинок, що демонструє його надійність для моделювання складної динаміки рідин в інженерії та комп'ютерній графіці.

Ключові слова: гідродинаміка згладжених частинок, SPH, симуляція рідини, Лагранжеві методи, нестисливість, функції ядра, пошук сусідів.