On the use of Particle-in-Cell methods in Visual Effects

Richard Southern*
National Center of Computer Animation
Bournemouth University

December 13, 2018

1 Introduction

After rendering, fluid simulation remains the most computationally expensive task in Visual Effects production. However turbulent and photorealistic fluids are arguably some of the most visually exciting dynamic physical phenomena used in film and are ubiquitous in new films. These effects are typically the result of multiple interacting visual components, the combination of which delivers a seemingly complex visual effect. Components (display in Fig. 1 may include

- Ocean waves populating the background of the shot,
- Main or "hero" waves in the foreground with potentially more complex behaviour,
- Droplets and splashes on the peak of the wave and against the rocks,
- Mist and vapour,
- Foam and churn in front of the wave near the short, and
- Bubbles entrained in the liquid creating lighter areas beneath the surface.

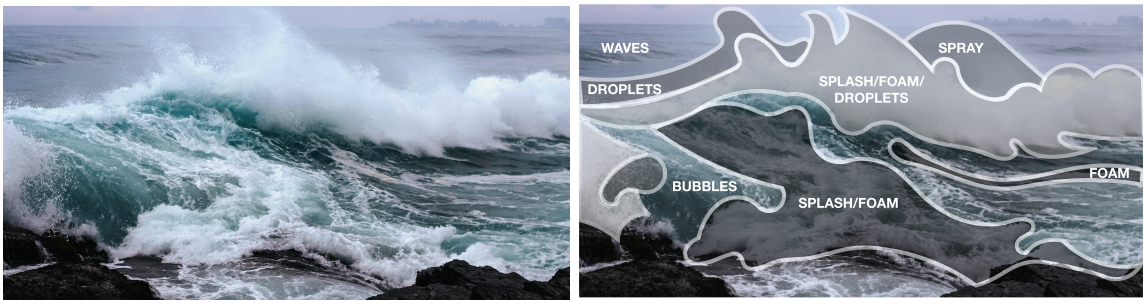

Figure 1: Wave phenomena consist of interactions between many different states of liquid and air mixtures including droplets, sprays, foam and bubbles, as well as larger scale liquid behaviours such as breaking waves and ocean wave dynamics. Image courtesy of Richard Jones [14].

Common practice for artists in all areas of Visual Effects is *layering*: the more physically plausible components that can be compiled into a shot the less likely the viewer will observe something implausible. This "bells and whistles" principle is demonstrated in [14], with an example shot from *American Assassin* (2017) requiring at least 78 different layers which are composited in the final rendered image. An example is included in Fig. 2 visually illustrate the process.

*rsouthern@bournemouth.ac.uk

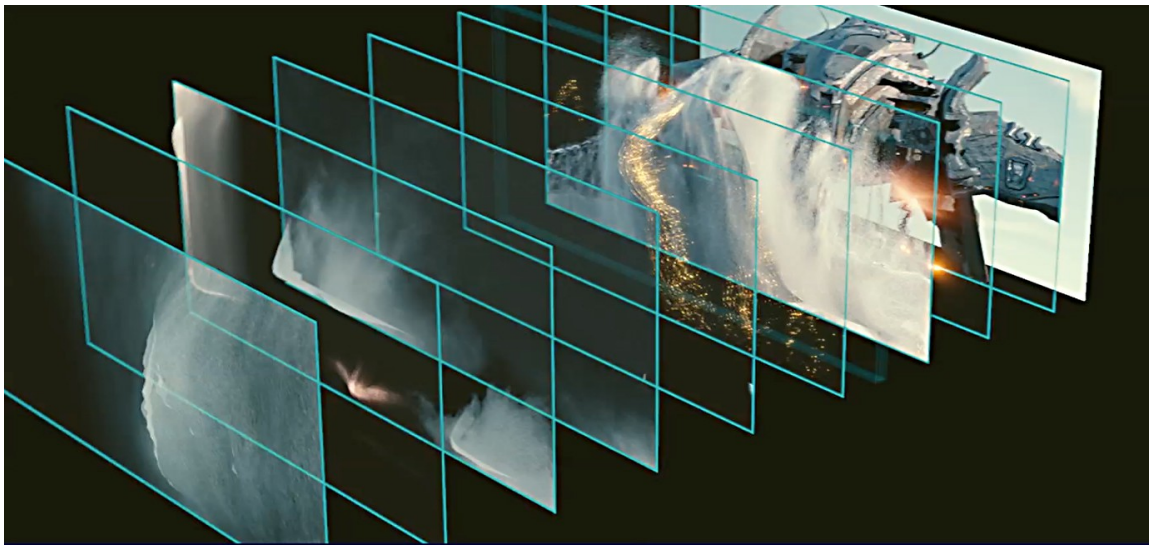

Figure 2: Typically a Visual Effects artist will simulate and partially control multiple different layers to construct the final shot. Image from FXGuide [9] from the film *Battleship*.

What is noteworthy is that many of these components are simulated using different approaches — the general ocean state is often modelled based on 2D heightfields (using a method like that of Tessendorf [21], developed while working at Rhythm and Hues) while spray and droplets may be modelled using ballistic methods. However the industry standard for foreground, high quality fluid simulation and associated effects is the Particle-In-Cell method, which I will describe here in some detail. Note that I will avoid discussion of the extremely important *boundary conditions* (the reader is referred to Bridson [5] for more details) and will only address implementation details at a high level.

2 Particle-In-Cell Methods

Francis Harlow is widely credited as being responsible for the establishment of computational fluid dynamics (CFD) as an important discipline. His work while at the Los Alamos National Laboratory, where he served as group leader for 14 years, has produced some of the most fundamental contributions to the area of CFD for computer simulation of fluids, including the Particle-In-Cell (PIC) and Marker-and-Cell (MAC) grid methods, which underpin the methods used in Visual Effects today.

Francis Harlow, Jan. 22, 1928 — July 1, 2016

The Particle-In-Cell (PIC) method was first introduced by Harlow and Welch [11]: their approach was one of the first proposed for numerical fluid dynamics, and was applied to the simulation of supersonic flows with obstacles [10]. In his words:

> One of the things about supersonic flows that I soon learned was that there were at that time two main ways to think about zoning space for resolving the behavior of a fluid. One of them we call the Lagrangian approach, which means having a mesh of computational cells that follow the motion of a fluid through its contortions and whatever processes that take place. The mesh could follow interfaces beautifully. The other, the Eulerian viewpoint, has a fixed mesh of cells that stay in one place in the laboratory frame and the fluid flows through it. This second approach is great for looking at large distortions; there is no movable computational mesh to get tangled. But on the other hand, it has problems if you want to follow a sharp interface between two fluids. Eulerian techniques tend to diffuse the interface and to smear out sharp shocks. (In

those days we had only begun to experiment with very primitive interface reconstruction techniques.) *Harlow [10]*

Many contemporary advances in fluid simulation applied to the domain of Visual Effects are attributed to Robert Bridson, and the crucial theory which underpins much of this talk is contained within his book [5]. The contemporary context and examples of usage are derived from conversations with and reference to Jones [14].

The central premise of PIC methods is to represent fluid as a hybrid representation (see Fig. 3), with particles and associated information where advection is computed, and a grid which computes everything else. The finite differencing approach to solving the Navier-Stokes equations has been updated since Harlow and Welch [11] and Brackbill et al. [4] to enforce stable incompressability based on the Poisson method of Stam [19], but contemporary methods differ in the manner in which particles are transfered to and from the underlying grid representation, and what information is stored with the particles. For the implementation, most methods still employ the original Marker–and–Cell (MAC) data structure of Harlow and Welch [11], but the scale of contemporary simulations require more sophisticated volumetric databases such as the Volumetric Database introduced by Museth [16].

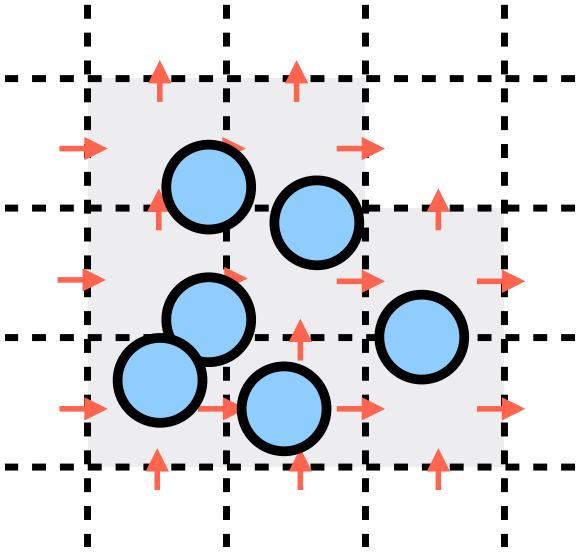

Figure 3: Particle-In-Cell methods are hybrid representations of fluid, with particles embedded in a grid representation. Advection is resolved on the particle representation, everything else is evaluated on the grid.

3 Solving the Navier-Stokes Equations with PIC

The Navier-Stokes equation is typically defined as

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= -\frac{\nabla p}{\rho} + \mathbf{g} + \nu \nabla^2 \mathbf{u} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \tag{1}$$

where $\mathbf{u}$ is the fluid velocity vector, p is the fluid pressure, ρ is the fluid density, ν is the kinematic viscosity coefficient, and ∇ and ∇^2 represent the gradient differential and Laplacian operators respectively.

Standard practice in VFX is to simply drop viscosity and use the Euler equations:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= -\frac{\nabla p}{\rho} + \mathbf{g} \\ \nabla \cdot \mathbf{u} &= 0 \end{aligned} \tag{2}$$

> In some situations, viscosity forces are extremely important: e.g., simulating honey or very small-scale fluid flows. But in most other cases that we wish to animate, viscosity plays a minor role, and thus we often drop it: the simpler the equations are, the better. In fact, most numerical methods for simulating fluids unavoidably introduce errors that can be physically reinterpreted as viscosity (more on this later) — so even if we drop viscosity in the equations, we will still get something that looks like it. In fact, one of the big challenges in computational fluid dynamics is avoiding this spurious viscous error as much as possible.
>
> *Bridson [5]*

For completeness, methods to evaluate viscosity for particle-in-cell methods are included in Appendix B.

Note also that

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \frac{D\mathbf{u}}{Dt}$$

is the material derivative, which allows the Euler equation to be written in a more compact form.

3.1 PDE Splitting

The first step is to define a simple forward Euler integration by splitting Eq. 2 into separate components. This method is inspired by the approach proposed by Stam [19] and is crucial to enforcing stability. For a separable differential equation

$$\frac{dq}{dt} = f(q) + g(q)$$

we can split this with Euler integration to read:

$$\tilde{q} = q^n + \Delta t f(q^n) \tag{3}$$

$$q^{n+1} = \tilde{q} + \Delta t g(\tilde{q}) \tag{4}$$

which can be shown to be first–order accurate (see Appendix C).

The Euler equations can now be divided into three effective steps using the method of splitting described above, and are typically applied in the following order:

1. The application of **body forces**:

   $$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{g}.$$

   This is the most straightforward step, which is solved according to the forward Euler method $\tilde{\mathbf{u}} = \mathbf{u} + \Delta t \mathbf{g}$, and this is typically computed on the projected positions of the Marker-and-Cell (MAC) grid (see below).

2. Solving for the **pressure projection** while preserving **incompressibility**:

   $$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho}\Delta p \tag{5}$$

   subject to $\nabla \cdot \mathbf{u} = 0$. This is somewhat harder to solve, and is covered in Section 3.3.

3. **Advecting**[1] the properties due to the velocity field:

   $$\frac{D\mathbf{u}}{Dt} = 0. \tag{6}$$

   Advection is performed on the particles, and the method of advection is dependent on the integration scheme. This can be written out as a PDE and solved using one of any number of integration methods. For example, a simple forward Euler advection step to update particle positions would be computed by

   $$x_p^{n+1} = x_p + \Delta t \mathbf{u}_p \tag{7}$$

   where the subscript p is used to indicate that these properties are updated on the particle representation.

Note that the crucial step of transfering particle properties to and from the grid representation is discussed later in Section 3.4.

[1] Note that while any properties may be advected by the velocity field, generally this is the particle positions x.

3.2 Discretization

Properties from the particles are transfered to points on a grid in order to solve Eq. 24 using partial differencing. For the method in [8] this entails the projection of pressure to cell centres (e.g. $p_{i,j}$) and particle velocities are projected to the centre of cell faces (e.g. $u_{i+1/2,j}$) such that neighbouring cells shared points (e.g. $v_{i,j-1/2,k} \equiv v_{i,(j-1)+1/2,k}$). This requires a staggered grid representation, as shown in Fig. 4.

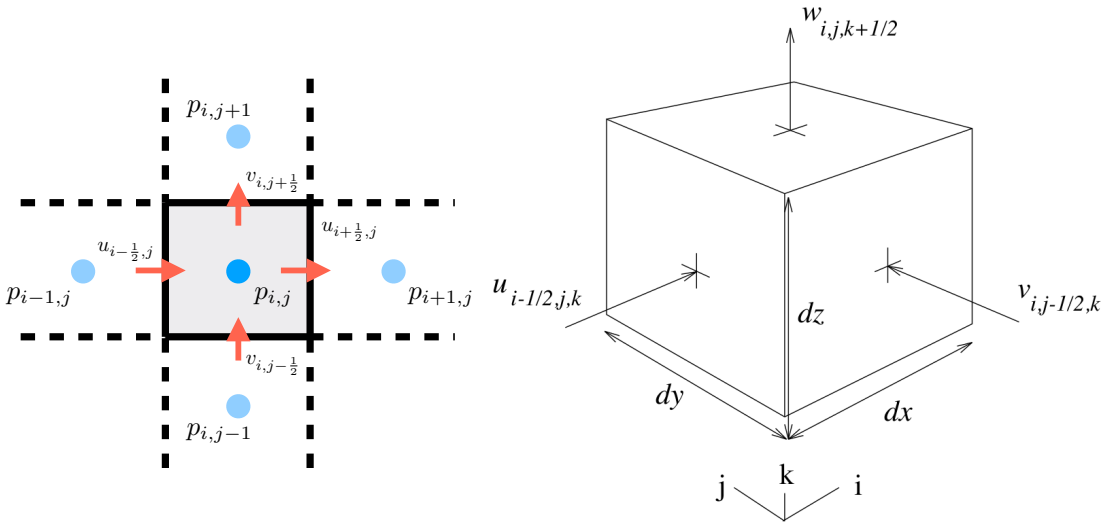

Figure 4: The Marker-In-Cell (MAC) grid in $\mathbb{R}^2$ on the left (from [14]) and in $\mathbb{R}^3$ on the right (from [8]). This structure was first introduced by Harlow and Welch [11].

Most methods use the fastest approach to determining weights for transfering propeprties from particles to the grid — which is to use *trilinear interpolation* (or *bilinear interpolation* in $\mathbb{R}^2$), a fast area weighting interpolation scheme[2]. The main advantage of the grid representation is that is allows the use of finite differencing to estimate quantities via second order accurate differences, e.g. at grid cell (i, j) with width Δx in $\mathbb{R}^2$:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \\ &\approx \frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{\Delta x} \end{aligned} \tag{8}$$

and similarly at the centre of face $(i - 1/2, j)$,

$$\frac{\partial p}{\partial x} \approx \frac{p_{i,j} - p_{i-1,j}}{\Delta x} \tag{9}$$

3.3 Pressure Projection

At the heart of PIC and the derived methods is the approach to subtract the pressure gradient from the intermediate velocity field $\mathbf{u}$ to ensuring incompressibility and enforce boundary conditions. Note that the approach presented here is the method commonly applied in Computer Graphics but differs from the original approaches from Harlow and Welch [11], Brackbill et al. [4] and Foster and Metaxas [8]. The pressure projection step in Eq. 5 can be expressed as:

$$\mathbf{u}^{n+1} = \mathbf{u} - \frac{\Delta t}{\rho} \nabla p \tag{10}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0 \tag{11}$$

[2]So common, it's on Wikipedia: https://en.wikipedia.org/wiki/Trilinear_interpolation.

In the discrete sense, Eq. 10 can be written as:

$$\begin{aligned} u^{n+1}_{i+1/2,j} &= u_{i+1/2,j} - \frac{\Delta t}{\rho}\frac{p_{i+1,j} - p_{i,j}}{\Delta x} \\ v^{n+1}_{i+1/2,j} &= v_{i,j+1/2} - \frac{\Delta t}{\rho}\frac{p_{i,j+1} - p_{i,j}}{\Delta x} \end{aligned} \tag{12}$$

while Eq. 11 can be written as

$$\nabla \cdot \mathbf{u}^{n+1} = \frac{u^{n+1}_{i+1/2,j} - u^{n+1}_{i-1/2,j}}{\Delta x} + \frac{v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2}}{\Delta x} = 0 \tag{13}$$

Substituting Eq. 12 into Eq. 13 and simplifying yields:

$$\begin{aligned} \frac{\Delta t}{\rho}\left(\frac{4p_{i,j} - p_{i+1,j} - p_{i,j+1} - p_{i-1,j} - p_{i,j-1}}{\Delta x^2}\right) = \\ -\left(\frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{\Delta x}\right) \end{aligned} \tag{14}$$

The system above is essentially the Poisson problem, and can be reformulated as the system $\mathbf{Ap} = \mathbf{b}$ where $\mathbf{A}$ is a 5 (in $\mathbb{R}^2$) or 7 (in $\mathbb{R}^3$) point Laplacian matrix, $\mathbf{p}$ is the solution vector of pressures, and $\mathbf{b}$ is the negative vector of divergence values at each cell.

The matrix $\mathbf{A}$ is sparse, symmetric and positive definite, meaning that it is easily solved using Preconditioned Conjugate Gradient or Multi-Grid approaches.

The solution to the above equation is then used to remove the divergence from the velocity field:

$$\hat{\mathbf{u}} = \tilde{\mathbf{u}} - \nabla p. \tag{15}$$

3.4 Property transfer to and from the grid

At each time step, the mass and momentum from each particle is transfered to the centre of faces on the MAC grid. In the original method of Harlow and Welch [11] the transfer from particles to the MAC grid is

$$m^n_i = \sum_p w^n_{ip} m_p \tag{16}$$

$$m^n_i \mathbf{u}^n_i = \sum_p w^n_{ip} m_p \mathbf{u}^n_p \tag{17}$$

where the subscript p indicates a particle property and i indicates a particular grid cell. m^n is mass at time step n (although for mass conservation this never changes), $\mathbf{u}^n$ is velocity at time step n, and w^n_{ip} is the interpolation weights associating the particle position x_p with the grid point x_i, deduced using bilinear (2D) or trilinear (3D) interpolation.

Transfering the updated velocities to the particles is achieved with

$$\mathbf{u}^{n+1}_{p,\text{PIC}} = \sum_i w^n_{ip} \mathbf{u}^{n+1}_i \tag{18}$$

The principle improvement offered by [4] was to change the backwards transfer by updating the original particle velocity by the difference rather than overwriting the velocity stored at particles, e.g.

$$\mathbf{u}^{n+1}_{p,\text{FLIP}} = \mathbf{u}^n_p + \sum_i w^n_{ip}(\mathbf{u}^{n+1}_i - \mathbf{u}^n_i). \tag{19}$$

This relatively minor adjustment to the algorithm greatly improves the quality of simulation results by almost eliminating the energy dissipation resulting from transfering to and from the grid representation.

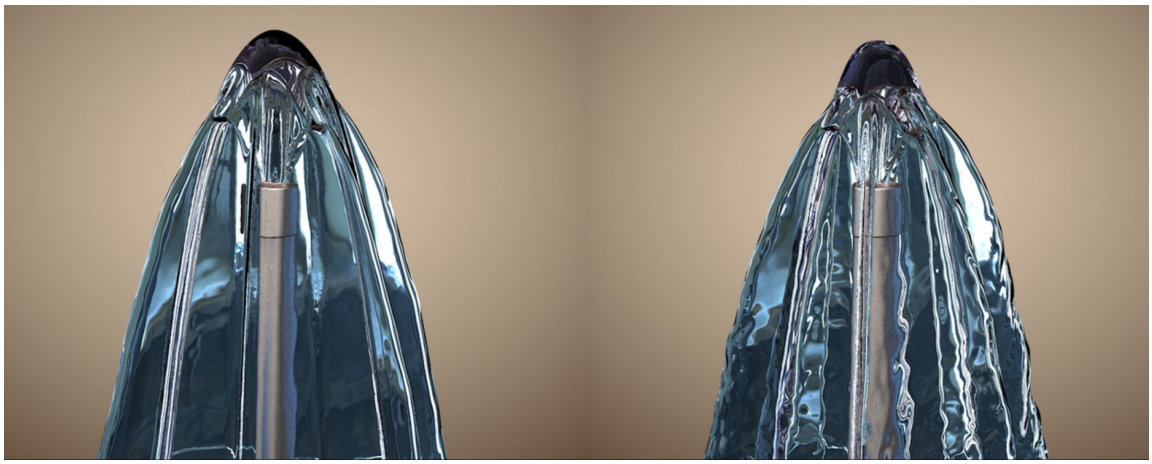

Figure 5: A comparison if APIC and FLIP/PIC (from [7]). APIC appears considerably smoother, with the FLIP/PIC solution generates noise — which may not be unwanted depending on the example.

FLIP simulations alone tend to be too "splashy", while PIC simulations tend not to be splashy enough due to the energy dissipation. Zhu and Bridson [22] proposed the linear of the results of both methods, e.g.

$$\mathbf{u}_p^{n+1} = (1-\alpha)\mathbf{u}_{p,\text{PIC}}^{n+1} + \alpha\mathbf{u}_{p,\text{FLIP}}^{n+1} \tag{20}$$

exposing a simple user parameter α which controls the "splashiness" of the fluid, although with little physical justification[3].

While linear momentum is conserved by both forward and backward transfers when using either FLIP or PIC, the angular momentum, quantified on either the grid or particles as

$$L_{\text{tot}} = \sum_i x \times m\mathbf{u} \tag{21}$$

is not preserved when transfering properties from the grid to the particles (Eq. 18) and is lost.

The solution offered by Jiang et al. [13] (working with Disney) is to store an additional two (2D) or three (3D) vectors at the particles $\mathbf{c}_{pa}$, where a represents the x, y, z direction. The transfer from particles to the grid for the mass is as in Eq. 16, but to the faces is then achieved using:

$$m_{ai}^n u_{ai}^n = \sum_p m_p w_{aip}^n \left(\mathbf{e}_a^T \mathbf{u}_p^n + (\mathbf{c}_{pa}^n)^T (x_{ai} - x_p^n)\right) \tag{22}$$

where $\mathbf{e}_a^T$ is the basis vector for axis a, and x_{ai} is the a-position of the grid node i. This process ensures that the component of angular momentum is incorporated in the grid velocity. The transfer of velocity from faces to particles is achieved using Eq. 18, but the vectors $\mathbf{c}_{pa}$ are updated using:

$$\mathbf{c}_{pa}^{n+1} = \sum_i \nabla w_{aip}^n \mathbf{u}_{ai}^{n+1}. \tag{23}$$

Note that this approach requires very little additional storage, and the performance overhead is minimal. The qualitative improvements are demonstrated in Fig 5.

While APIC significantly improves on the smoothness of the results by preserving the angular momentum and to date is incorporated into nearly every content creation tool, FLIP/PIC is still used in some circumstances by VFX artists due to the additional control they have over damping (the parameter α in Eq.20) and the splashy, noisy effect that emerges naturally (see, for example Fig. 5). There "swirly" effects arising from APIC as a result of the preservation of angular momentum may also be undersirable.

[3] Bridson [5] derives an explicit value for α from the numerical dissipation implied by PIC, based on the kinematic viscosity of fluid ν: $\alpha = \min\left(\frac{6\Delta t\nu}{\Delta x^2}, 1\right)$.

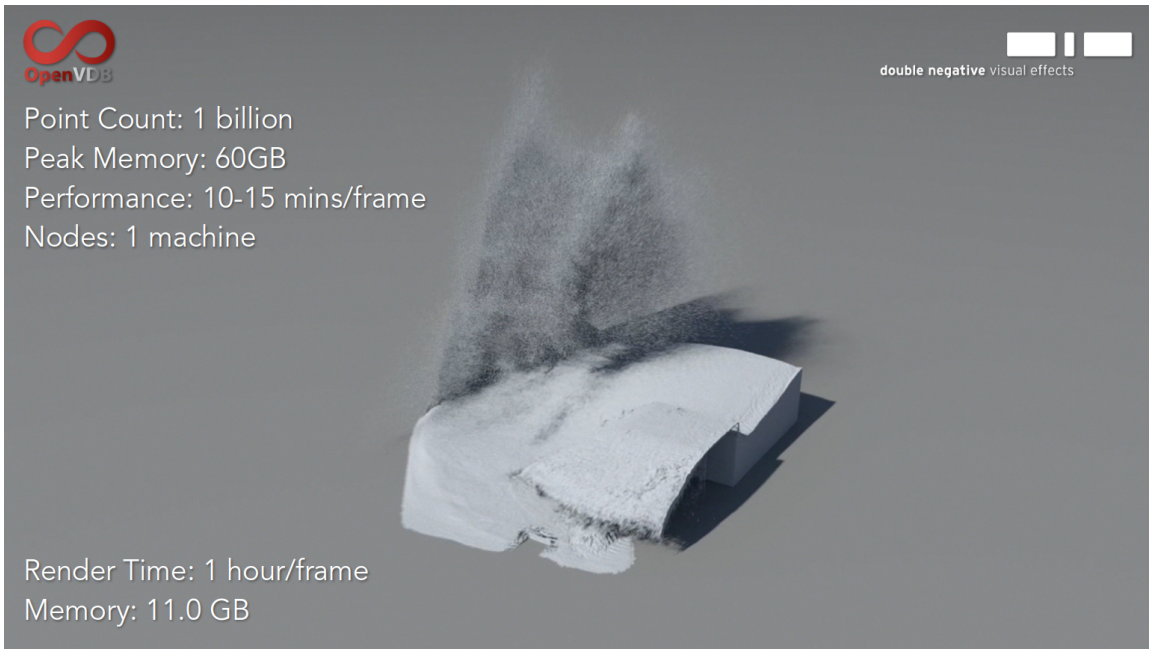

Figure 6: Some performance metrics for large scale FLIP simulation in VFX production using OpenVDB. From Baily [2].

4 Implementation

The Volumetric Database (VDB) was first introduced by Museth [16] for use within DreamWorks Animation. This tool has proven to be exceptionally versatile for a number of applications in film production and computer graphics, winning the Sci-Tech Academy Award in 2014. The VDB stores near infinite volumes of 3D data in cache–coherent, rapid access structure with $O(1)$ random access for insertion, retrieval and deletion. This is supported through an adaptive hierarchical structure, with a quantised representation of local data allowing for orders of magnitude reductions in storage overhead.

Amongst the many application domains within Computer Graphics that this technology has been used, Museth [16] specifically discusses the use of this approach in the context of PIC methods:

> "In the more practical context of surface tracking in fluid simulation, level set advection often involves velocity fields represented on MAC grids. These "staggered grids" conceptually store the Cartesian components of the velocities at cell faces rather than colocated at cell centers. This is a popular technique to suppress so-called checkerboard frequencies in the pressure projection step of most Navier-Stokes solvers. However, instead of devising specialized data structures for MAC grids, it is customary to use standard collocated grids, and simply move the staggered interpretations of the grid indexing and values to client code. In other words, VDB fully supports MAC grids, and we use special interpolation schemes to convert from staggered to colocated velocities required for level set advection." *Museth [16]*

An extension was developed and maintained by DNEG [6] to specialise the VDB representation to store points and implement fast routines for MAC-grid projection, which has been widely adopted and used in the context of Visual Effects production. An example of the memory overhead for large simulations is shown in Fig. 6, with billion particle simulations possible on a standard workstation. A significant advantage of this representation (and FLIP in general) are the use of fast surfacing techniques which are able to generate signed distance field representations which are necessary for fast rendering of fluid volumes.

Distributed simulations are crucial for high performance simulation, and typical implementations exploit CPU (rather than GPU) parallelism due to the significant memory requirements. The most expensive operation of the FLIP solve is the pressure projection step (outlined in Section 3.3) which

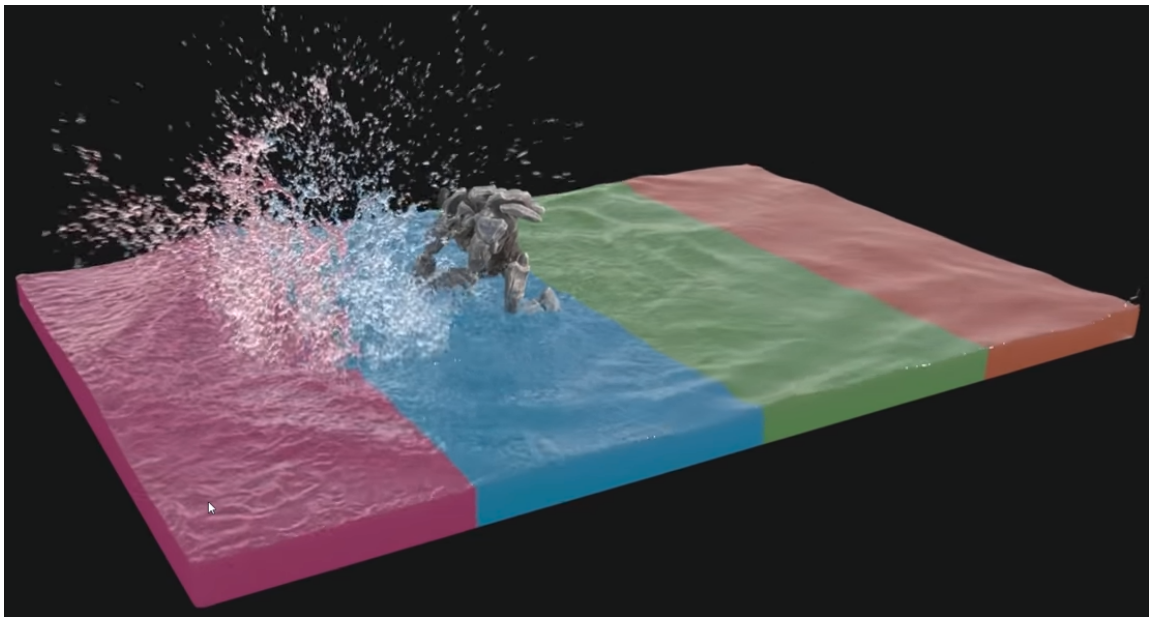

Figure 7: An example of distributed FLIP, as implemented in Houdini 15. Image from SideFX [18].

requires the solution of a sparse linear system. This is solved in a distributed fashion through the use of a highly specialised solver which takes advantage of the spatial organisation and the small size of the Laplacian stencil. A distributed framework for solving FLIP in a production environment is discussed briefly in Bailey et al. [1]. In the toy example in Fig. 7 it is clear that distributed simulations do not suffer from artefacts across boundaries between cells.

5 FLIP in Content Creation Tools

Insights on PIC methods used in VFX are provided by Jones [14]:

- Whilst simulations are comparatively fast, they are dominated by the time to perform the pressure projection (Section 3.3).
- Interaction and fluid behaviour happens on the scale of the background grid, so whilst particles are present in the model they do not represent any particular liquid mass.
- As the particles do not interact with each other directly, they are not guaranteed to be well-distributed.
- The simulations require manipulation of both point and volumetric data, and so artists may require ways to interact with both of these representations.

> As well as the computational benefits of this method, there are workflow implications that have made it so successful. Firstly, particle systems are a very familiar concept to FX artists and so having particles as the input and output of the simulation makes it very easy to work with in a VFX environment. Building on this, FLIP/PIC in particular facilitates particle workflows that are much more forgiving than purely Lagrangian methods such as SPH.
>
> This method allows the user to arbitrarily reseed the particle set under the fluid surface and introduce additional particles without destabilising the simulation whilst retaining benefits of particles such as fine details at the surface (at sub-grid resolution) and handling of splashes that would be difficult to resolve on a grid representation. From this it is not difficult to see why it has become a favourite of visual effects artists. *Jones [14]*

Realflow implements FLIP as part of it's *Hybrido* toolset:

> Like PBD methods, FLIP solvers do not need a high number of substeps. Low substeps makes these solvers very fast, but this can also be a problem in terms of fluid-object interaction. RealFlow provides a wide variety of solution to fix these interaction problems. In FLIP solvers, the particles are used to carry the fluid's velocity information. During the simulation, velocity is transferred to the grid, where the physics is done. This transfer can be seen as the core that keeps the simulation stable. The main advantage of the hybrid approach is, aside from speed, that it is possible to add secondary fluids like splashes, foam, or mist.
>
> *RealFlow [17]*

Houdini has support for both FLIP/PIC and SPH and provides the following description for the comparison of the two methods:

> FLIP fluids are faster than SPH fluids, if you don't need to substep the FLIP fluid. If you need to substep the FLIP fluid because of fast moving colliders, you may find SPH just as a fast or faster. FLIP fluids are also useful because particles can be placed on top of each other without destabilizing the system. SPH tends to blow up if you move particles too close.
>
> The advantage of the FLIP Solver is that you run with only a few time steps per frame while SPH requires anywhere from 7 to 20 time steps or more per frame to stabilize. FLIP utilizes a few grids (volume fields) to help tame the instantaneous impulses that can arise in a fluid sim.
>
> Various fields are used to tame the FLIP Solver so that you can run far fewer points at far fewer time steps and the inter-spacing between particles can be random. You can introduce new particles at any time with little to no consequence. This opens up so many new work flows in POPs that were simply not possible with SPH. For example, introducing splash particles with their own property attributes is now possible. *Houdini [12]*

Houdini refers to FLIP as "splashy" and APIC as "swirly" to assist artists in identifying the appropriate tool. It is also commonly observed that due to the conservation of inertial properties APIC surfaces appear considerably smoother than FLIP (see Fig 5). The noisiness of FLIP could (counterintuitively) be considered a strength: artists often find it more suitable for generating agitated fluid than applying a noisy vector field to an APIC solution.

6 Fluid Problems in Visual Effects

DNeg is the multi-award winning visual effects studio behind Oscar winning films like Inception, Interstellar and Ex Machina. I asked them to identify some of the open challenges that they are encountering in fluid simulation for Visual Effects. They are listed below in priority order.

1. **Coupling of fluid solver with 2D ocean surface simulation**: Example given: a tiny dingy is being simulated in an open ocean. It is impractical (and uncontrollable) to simulate the entire ocean with a full fluid solver. The wider ocean is solved using a 2D (height field) simulation of the ocean surface, while the area about the boat is simulated using a full particle based fluid solver. Linking these two so that the resulting simulation is seamless remains an open problem. This links closely to the existing SPH grand challenge of coupling between existing fluid methods.

2. **Fluid authoring tools**: A very challenging open problem is how to provide artistic control for fluid in visual effects. There is fundamentally a tradeoff between a simulation that is entirely physically plausible and one that is largely artistically controlled, and typically the amount of control varies depending on the specific shot. Artists tend to prefer drawn images, so the ability to map a fluid state from a sketch would be a powerful and popular tool. The ability to determine the conditions necessary to transition between explicitly defined keyframes is also an open problem. From MW's perspective, he is sceptical that a unifying general purpose tool can be created that satisfies all artistic requirements on all shots required in VFX.

3. **Adaptibility / multiresolution**: There is no dispute that more particles leads to generally more believable physical simulations of fluid. DNeg, one of the leaders in fluid simulation in Visual Effects, can currently simulate systems of about 3 billion particles using their distributed FLIP solver (which was needed for the massive wave in Interstellar). Demand for more particles will continue to grow, but processing and storage requirements will increase exponentially. Currently there is a requirement that fluid data can be visualised on a high end workstation, which limits the maximum resolution of any resulting fluid simulation. Multi-resolution solutions (already one of the grand challenges) which allow resolution to be adaptively refined based on regions of interest would reduce hardware requirements significantly.

4. **Multiphase simulation**: Multiphase fluid simulation is a widely relevant problem, and also crops up sometimes in visual effects. A particularly common problem is the simulation of foam on ocean waves, but there are other examples of miscible and immiscible multi-phase fluid behaviours which do occasionally need to be simulated.

Current solutions to these problems are inaccurate, expensive to compute, or do not offer sufficient artistic control, and would benefit from research.

7 Concluding Remarks

The widespread use of Particle–In–Cell methods in Visual Effects is a testament to the ingenuity of early works by Harlow and coworkers so many decades ago. The performance, stability and distributable nature make it the tool of choice for use in the Visual Effects industries. It should also be noted that the PIC approach has been applied to other simulation problems, such as the Material Point Method — most notably used in Disney's *Frozen* [20], and research continues in this area to simulate more complex material behaviours.

References

[1] Dan Bailey, Harry Biddle, Nick Avramoussis, and Matthew Warner. Distributing liquids using openvdb. In *ACM SIGGRAPH 2015 Talks*, SIGGRAPH '15, pages 44:1–44:1, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3636-9. doi: 10.1145/2775280.2792544. URL http://doi.acm.org/10.1145/2775280.2792544.

[2] Dan Baily. OpenVDB course: Advanced applications of openvdb in production. URL http://www.openvdb.org/download/openvdb_particle_storage_2015.pdf.

[3] Christopher Batty and Robert Bridson. Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 219–228, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association. ISBN 978-3-905674-10-1. URL http://dl.acm.org/citation.cfm?id=1632592.1632624.

[4] J.U. Brackbill, D.B. Kothe, and H.M. Ruppel. Flip: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25 – 38, 1988. ISSN 0010-4655. doi: https://doi.org/10.1016/0010-4655(88)90020-3. URL http://www.sciencedirect.com/science/article/pii/0010465588900203.

[5] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

[6] DNEG. Openvdb points. URL http://www.openvdb.org/documentation/doxygen/points.html.

[7] Effex. Effex 2.8 teaser 1: New APIC solver. URL https://vimeo.com/183282573.

[8] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471 – 483, 1996. ISSN 1077-3169. doi: https://doi.org/10.1006/gmip.1996.0039. URL http://www.sciencedirect.com/science/article/pii/S1077316996900398.

[9] FXGuide. Battleship: tactical water and fluid sims. URL https://www.fxguide.com/featured/battleship-tactical-water-and-fluid-sims/.

[10] Francis H. Harlow. Fluid dynamics in Group T-3 Los Alamos National Laboratory: (LA-UR-03-3852). *Journal of Computational Physics*, 195(2):414 – 433, 2004. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2003.09.031. URL http://www.sciencedirect.com/science/article/pii/S0021999103005692.

[11] Francis H. Harlow and J. Eddie Welch. Numerical Calculation of Time–Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The Physics of Fluids*, 8(12):2182–2189, 1965. doi: 10.1063/1.1761178. URL https://aip.scitation.org/doi/abs/10.1063/1.1761178.

[12] SideFX Houdini. FLIP solver. URL http://www.sidefx.com/docs/houdini/nodes/dop/flipsolver.html.

[13] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 34(4):51, 2015. URL https://dl.acm.org/citation.cfm?id=2766996.

[14] Richard Jones. *Droplets, Splashes and Sprays: Turbulent Liquids in Visual Effects Production*. PhD thesis, Bournemouth University, 2019.

[15] Egor Larionov, Christopher Batty, and Robert Bridson. Variational stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Trans. Graph.*, 36(4):101:1–101:11, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073628. URL http://doi.acm.org/10.1145/3072959.3073628.

[16] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.*, 32(3):27:1–27:22, July 2013. ISSN 0730-0301. doi: 10.1145/2487228.2487235. URL http://doi.acm.org/10.1145/2487228.2487235.

[17] RealFlow. Glossary: RealflowâĂŹs liquid solver types. URL https://blog.realflow.com/technology/glossary-realflows-liquid-solver-types/.

[18] SideFX. Houdini 15 masterclass. URL https://www.youtube.com/watch?v=J86ycHU_ppc.

[19] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5. doi: 10.1145/311535.311548. URL http://dx.doi.org/10.1145/311535.311548.

[20] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Trans. Graph.*, 32(4):102:1–102:10, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2461948. URL http://doi.acm.org/10.1145/2461912.2461948.

[21] Jerry Tessendorf. Simulating ocean waves. URL https://people.cs.clemson.edu/~jtessen/papers_files/coursenotes2004.pdf.

[22] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, July 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073298. URL http://doi.acm.org/10.1145/1073204.1073298.

A Older method resolve FLIP/PIC pressure projection

The method described by [8] uses a relaxation method to eliminate velocity diffusion, which I include here for completeness. Eq. 1 can be decomposed into it's partial derivatives in each of the component dimensions (in $\mathbb{R}^2$):

$$\begin{aligned}\frac{\partial u}{\partial t}+\frac{\partial u^2}{\partial x}+\frac{\partial uv}{\partial y}&=-\frac{\partial p}{\partial x}+g_x+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)\\ \frac{\partial v}{\partial t}+\frac{\partial vu}{\partial x}+\frac{\partial v^2}{\partial y}&=-\frac{\partial p}{\partial y}+g_y+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right),\end{aligned} \tag{24}$$

where u, v are velocities in the x, y directions respectively, p is the local pressure, g is gravity and ν is the kinematic viscosity of the fluid. Terms on the left hand side of the equations account for changes in velocity due to local fluid acceleration and convection. The right hand terms take account of acceleration due to the force of gravity (or any body force g), acceleration due to the local pressure gradient (∇p) and drag due to kinematic viscosity (ν) or thickness of the fluid [8].

In Harlow and Welch [11], Brackbill et al. [4] and Foster and Metaxas [8], Eq.24 is solved by finite differencing on the discretization in Fig 4:

$$\begin{aligned}u^{n+1}_{i+1/2,j} =& u_{i+1/2,j}+\Delta t\{(1/\Delta x)[u^2_{i,j}-u^2_{i+1,j}]\\ &+(1/\Delta y)[(uv)_{i+1/2,j-1/2}-(uv)_{i+1/2,j+1/2}]+g_x\\ &+(1/\Delta x)(p_{i,j}-p_{i+1,j})\\ &+(\nu/\Delta x^2)(u_{i+3/2,j}-2u_{i+1/2,j}+u_{i-1/2,j})\\ &+(\nu/\Delta y^2)(u_{i+1/2,j+1}-2u_{i+1/2,j}+u_{i+1/2,j-1}),\end{aligned} \tag{25}$$

yielding an explicit approximation of the updated velocity $u^{n+1}_{i,j}$ in terms of the projected pressure at cell centres and the projected velocities at staggered grid points.

The result is not divergence free / incompressible — Foster and Metaxas [8] determine the updated pressure field by solving the mass conservation equation $\nabla\mathbf{u}=0$ by defining the fluid divergence (or "missing mass") for a cell (i,j) according to:

$$D_{i,j}=-\left(\frac{u_{i+1/2,j}-u_{i-1/2,j}}{\Delta x}+\frac{u_{i,j+1/2}-u_{i,j-1/2}}{\Delta y}\right) \tag{26}$$

with the change in pressure given by:

$$\Delta p_{i,j}=\frac{\beta_0}{2\Delta t}\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)D_{i,j} \tag{27}$$

where β_0 is a relaxation coefficient within the range $[1,2]$. The cell face velocities and cell pressure properties are then updated (see Foster and Metaxas [8] for the specifics), and the process is repeated. This iterative process converges in 3-6 iterations according to the paper.

B Viscosity

As stated previously, fluids are considered by default in VFX to be inviscid, but solutions to this are available. Batty and Bridson [3] solve viscosity as a separate step by the following PDE:

$$\hat{\mathbf{u}}=\frac{1}{\rho}\nabla\cdot\left(\nu\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right)\right) \tag{28}$$

which is discretized to give the following form:

$$\hat{\mathbf{u}}=\mathbf{u}+\frac{\Delta t}{\rho}\nabla\cdot\left(\nu\left(\nabla\mathbf{u}^*+(\nabla\mathbf{u}^\dagger)^T\right)\right) \tag{29}$$

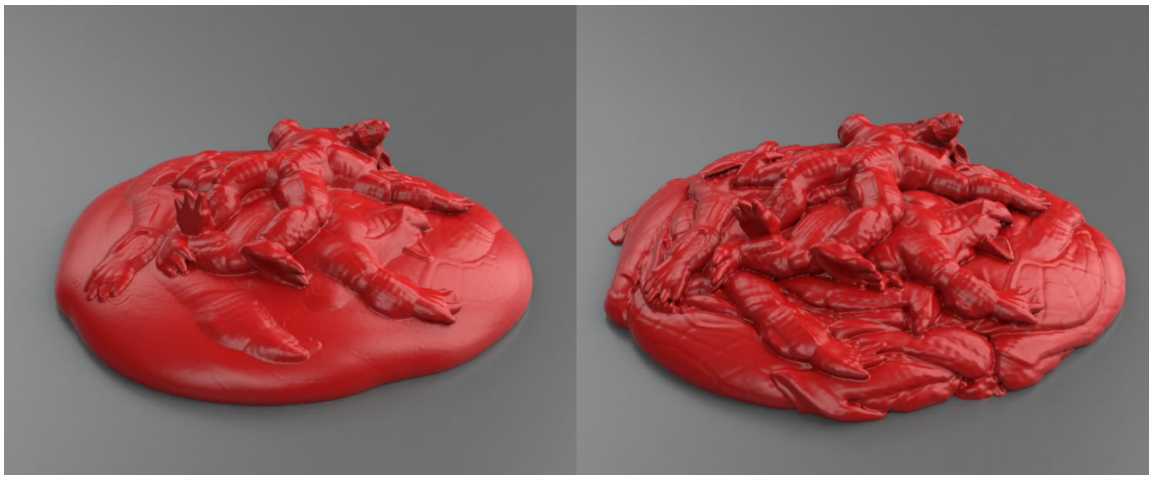

Figure 8: Collapsed piles of viscous armadillos. Boundary conditions of the method of Batty and Bridson [3] (left) reduce surface detail in comparison with the method of Larionov et al. [15]. Reproduced from Larionov et al. [15].

where $\mathbf{u}^*$ and $\mathbf{u}^\dagger$ are used to denote whether the old or current version of $\mathbf{u}$ are used depending on the integration scheme. Once discretised this solution adds an additional linear solve to the system described above.

Larionov et al. [15] combine the viscous and pressure force solve into a single coupled system, and are able to address limitations in the decoupled approach of Batty and Bridson [3] (see Fig. 8. However the additional computational cost is not currently considered worth the qualitative improvement.

C On the splitting of Ordinary Differential Equations

[5] demonstrate that ODE splitting is first order accurate using Taylor series expansion. Substituting Eq. 3 into Eq. 4 yields:

$$\begin{aligned} q^{n+1} =& (q^n + \Delta t f(q^n)) + \Delta t g(q^n + \Delta t f(q^n)) \\ =& q^n + \Delta t f(q^n) + \Delta t (g(q^n) + O(\Delta t)) \\ =& q^n + \Delta t (f(q^n) + g(q^n)) + O(\Delta t^2) \\ =& q^n + \frac{dq}{dt}\Delta t + O(\Delta t^2). \end{aligned}$$