Droplets, Splashes and Sprays: Highly Detailed Liquids in Visual Effects Production

Richard Jones
Doctor of Engineering in Digital Media
Bournemouth University
Centre for Digital Entertainment
Faculty of Media and Communication
September 2018

Supervisors
Dr Richard Southern
Ian Masters

Abstract

An often misunderstood or under-appreciated feature of the visual effects pipeline is the sheer quantity of components and layers that go into a single shot, or even, single effect. Liquids, often combining waves, splashes, droplets and sprays, are a particular example of this. Whilst there has been a huge amount of research on liquid simulation in the last decade or so, little has been successful in reducing the number of layers or elements required to create a plausible final liquid effect. Furthermore, the finer-scale phenomena of droplets and sprays, often introduced in this layered approach and crucial for plausibility, are some of the least well catered-for in the existing toolkit. In lieu of adequate tooling, creation of these elements relies heavily on non-physical methods, bespoke setups and artistic ingenuity.

This project explores physically-based methods for creating these phenomena, demonstrating improved levels of detail and plausibility over existing non-physical approaches. These provide an alternative to existing workflows that are heavily reliant on artistic input, allowing artists to focus efforts on creative direction rather than trying to recreate physical plausibility.

We explore various approaches to increasing the level of detail captured in physically-based liquid simulations, developing a collection of tools that improve existing workflows. First, we investigate the potential of a re-simulation approach to increasing artist iteration on fluid simulations using previous simulation data. Following this, a novel droplet interaction model for ballistic particle simulations is developed to introduce higher levels of detail in simulations of liquid droplets and sprays. This allows physically-plausible interactions between droplet particles to be modelled efficiently and helps to create realistic droplet and spray behaviours. Then, to maximise the quality of the results of these and other particle-based simulations, we develop a high quality particle surfacing algorithm to handle the varied nature of inputs common in production. Finally, we discuss the development of an expression language to manipulate point and volume data commonly used in creating these simulations, as well as elsewhere throughout visual effects.

This research was driven directly by production requirements in partnership with a world-leading visual effects studio, DNEG. Projects have been developed to immediately integrate into production, using efficient, industry-standard, open technologies such as OpenVDB. It is shown that the toolkit for splashing liquids, even at fine-scales, can be improved by incorporating greater physical motivation further demonstrating the importance of physical simulation in visual effects and suggesting effects similarly reliant on artistic input (e.g. character/skin deformation) may benefit from increased physical correctness.

Contents

List of Figures **vi**

List of Tables **ix**

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Academic Context 3
- 1.3 Industrial Context 3
  - 1.3.1 Visual Effects (VFX) 4
  - 1.3.2 Industrial Partner: DNEG 4
  - 1.3.3 Considerations for Industry-Focused Research 7
- 1.4 Key Challenges of VFX Production 7
- 1.5 Project Objectives 9
- 1.6 Contributions 9
  - 1.6.1 Publications 10
  - 1.6.2 Software 10
- 1.7 Thesis Outline 10

2 Liquid Simulation for VFX **12**
- 2.1 Fundamentals of Fluid Motion 12
- 2.2 Fluid Simulation for Computer Graphics 14
  - 2.2.1 Eulerian Methods 14
  - 2.2.2 Lagrangian Methods 16
  - 2.2.3 Hybrid Methods 16
  - 2.2.4 Summary 17
- 2.3 Fluid Implicit Particle (FLIP)/Particle-In-Cell (PIC) Simulation 17
  - 2.3.1 Algorithm Overview 19
- 2.4 Realistic Splashing Liquid Effects 23
  - 2.4.1 Anatomy of a Splash 23
  - 2.4.2 Previous Approaches 25
- 2.5 How to Create a Production Quality Liquid Effect 28
- 2.6 Areas for Improvement 31

3 Increasing Iteration: Fluid Re-Simulation using Model Reduction **33**

3.1 Overview . . . 33
3.2 Model Reduction . . . 34
3.2.1 Projection . . . 34
3.2.2 Related Work . . . 35
3.2.3 Proper Orthogonal Decomposition via Method of Snapshots . . . 36
3.3 Subspace Pressure Projection using Previous Simulation Data . . . 38
3.3.1 Method . . . 39
3.3.2 Results . . . 41
3.3.3 Analysis . . . 46
3.4 Improvements to Model Reduction for Re-Simulation . . . 47
3.4.1 Full Dimension Pressure Projection with a Reduced Preconditioner . . . 47
3.4.2 Divergent Basis . . . 48
3.4.3 Basis Interpolation Methods . . . 50
3.4.4 Fluid Re-Simulation using Subspace Condensation . . . 50
3.5 Summary . . . 52

4 Increasing Detail: Physically-Based Droplet Interaction 53
4.1 Overview . . . 53
4.2 Motivation . . . 53
4.3 Related Work . . . 54
4.3.1 Issues with Existing Droplet Methods . . . 56
4.4 Real Droplet Interactions . . . 58
4.4.1 Binary Droplet Collisions . . . 59
4.5 Droplet Collision Modelling . . . 60
4.5.1 Parametrising Droplet Collisions . . . 62
4.5.2 Collision Outcome Classification . . . 64
4.5.3 Post-Collision Characteristics . . . 66
4.5.4 Application to Graphics . . . 68
4.6 Our Approach . . . 69
4.6.1 Robust-to-Input Outcome Classification . . . 70
4.6.2 Visually Plausible Post-Collision Characteristics . . . 71
4.6.3 Binary Collision Detection . . . 73
4.7 Implementation . . . 76
4.8 Results . . . 80
4.8.1 Limitations . . . 81
4.9 Summary . . . 82

5 Preserving Detail: Surfacing of Splashes with Droplets 88
5.1 Overview . . . 88
5.2 Motivation . . . 88
5.3 Related Work . . . 89
5.4 Our Approach . . . 91
5.4.1 Robust Anisotropic Kernels for Particle Fluid Surfacing . . . 92
5.5 Implementation . . . 94

5.6 Results 94
5.7 Analysis 96
5.8 Summary 98

6 Increasing Control: Particle and Volume Manipulation with AX 101
6.1 Overview 101
6.2 Motivation 102
6.3 Portable and Domain-Specific Languages 103
6.4 Our Approach 104
6.4.1 Task-Specific and Parallel by Design 104
6.5 Implementation 105
6.5.1 LLVM Function Generation 105
6.5.2 OpenVDB Integration 105
6.5.3 Available Operations 106
6.6 Examples and Use in Production 106
6.6.1 Example 1: Simulation 106
6.6.2 Example 2: FX to Lighting 106
6.6.3 Case Study: Dropping Pills on Loro 108
6.6.4 Production Statistics 109
6.7 Summary 110

7 Conclusion & Future Work 112
7.1 Discussion 112
7.2 Future Work 114
7.2.1 Extending This Work 114
7.2.2 Other Open Problems in High Quality Liquid Recreation 115
7.3 Conclusion 115

References 118

Appendix: Collision Threshold Derivation 131
Stretching Separation 131
Reflexive Separation 132

List of Acronyms 134

List of Figures

1.1 Time spent on different stages of production (hands-on artist time) at DNEG on 6 productions from 2014-2016. Effects development takes around 14% of the time, second only to compositing (which is required for every shot of every film). It is our understanding that this remains representative of the work undertaken in 2018. Figure taken from Barber et al. [2016]. 5
1.2 R&D at DNEG develops tools to solve a large range of issues in VFX production. These include the development of solvers capable to handling massive data-sets such as the distributed Dynamo Liquid solver (left - image taken from Bailey et al. [2015]) and highly efficient ways of storing and using geometry data such as OpenVDB Points (right - image taken from Museth et al. [2015]). 6

2.1 The three main discretisations used in liquid simulation for computer graphics. Left to right - **Eulerian**, **Lagrangian** and **Hybrid**. 14
2.2 Standard particle and grid setup in FLIP/PIC/APIC simulations. 18
2.3 A 2-dimensional staggered MAC grid cell. 19
2.4 A wave breaking on the rocks (left) and a waterfall (right). Phenomena such as these are extremely complex, containing interactions between many different states of the liquid and air mixtures including droplets, sprays, foam, and bubbles, as well as larger scale liquid behaviours such as the wave breaking and the ocean wave dynamics. 24
2.5 Dissecting the breaking wave from Figure 2.4 into its visible fluid components. Highlighting areas of predominantly splashes, foam, droplets and sprays. Note how the shape of the wave breaking and other similar larger-scale fluid behaviours are obscured by the smaller-scale fluid components. 25
2.6 A shot from the recent film *American Assassin* [Cuesta 2017] demonstrating a liquid splash as a ship's mast crashes into the water. Effects by DNEG's Ole Eidsheim. ©2017 CBS FILMS INC. and LIONSGATE ENTERTAINMENT INC. 28
2.7 Work-in-progress renders and contact sheets of some of the rendered elements created to create the splash effect in Figure 2.6. These elements include extra simulation passes for whitewater (spray, foam, bubbles), mist and vapour, as well as both point and surface renderings of the initial FLIP fluid simulation. In total 78 unique elements were passed on to the next stage of the pipeline for this effect. . 29

3.1 Single frame of 128×128 smoke plume style simulation 37

3.2 The first 4 elements $\mathbf{x}_1, ..., \mathbf{x}_4$ of our pressure basis $\mathbf{X}$ for the 128^2 sim in Figure 3.1. 38
3.3 The first 4 elements $\mathbf{v}_1, ..., \mathbf{v}_4$ of our velocity basis $\mathbf{V}$ for the 128^2 sim in Figure 3.1. 38
3.4 Simulation of 128^2 smoke plume using reduced pressure solution (Equation 3.12) only. Demonstrating visible accumulation of divergence (left), with addition of 5 Jacobi iterations (right). 40
3.5 Comparisons of errors in simulations using subspace pressure solves against those using full pressure solves for the simulation in Figure 3.1. Using relative L_2-error. 43
3.6 Log-log plot of divergence in the velocity POM vs singular value of that POM for 128^2 smoke plume simulation. 44
3.7 Error and max divergence when varying the buoyancy parameter in the 128^2 Reduced Simulation using only Direct Projection. Buoyancy for the input simulation $b = 9.81$. 44
3.8 Final state of a simulation with buoyancy parameter $b = 20$. Full solve (left) and *Direct Projection* solve with $b = 20$ (right) where input basis simulation has value for $b = 9.81$. Note the large variation in the final state between these two simulations. 45

4.1 Standard particle-based droplet simulation techniques and their interaction mechanisms. In no existing method does a single particle meaningfully represent a mass of liquid such as a droplet. 57
4.2 Droplet collision from experiment: initial coalescence of interacting droplets followed by subsequent separation and fragmentation. Reproduced with permission from Brazier-Smith et al. [1972]. 59
4.3 Main droplet collision types (start - top, end - bottom); Left to right - **Coalescence**, **Stretching Separation** and **Reflexive Separation** (prolate phase). 61
4.4 Geometric collision parameters for colliding droplets i and j. 63
4.5 Interaction volumes of droplets i and j. ϕ_i, ϕ_j are the fractions of the droplet volumes that are overlapped as droplet j is swept along the velocity $\mathbf{u}_{ij}$. 65
4.6 Our collision thresholds (for $\delta = 1$) based on those of Ashgriz and Poo [1990], with reflexive separation taking precedence for low impact number collisions. 71
4.7 ϕ_j (Equation 4.11) against X for $\delta = 0.4$, note that as $X \to 0$, the volume fraction ϕ_j in fact reduces as the entire droplet should be overlapped i.e. $\phi_j = 1$. 72
4.8 Shattering reflexive separation (left to right), greatly increased kinetic energy version of reflexive separation seen in Figure 4.3, creating many satellites with perturbed satellite velocities (§4.6.2). 73
4.9 Simulated results of different droplet collision outcomes, using our method. 80
4.10 Stretching separation between droplets displayed by a FLIP simulation. Requires 18k particles to replicate behaviour that we simulate with a single pair of particles (Figure 4.9b). 80
4.11 Small-scale system of droplet interactions: close-up using our droplet collision model (top); zoomed out view using our model (middle left), using FLIP solver (middle right), Smoothed Particle Hydrodynamics (SPH) solver (bottom left) and Rigid Body (RBD) solver (bottom right). While FLIP does create larger-scale motions, other systems are unable to create realistic small-scale droplet characteristics, e.g. size variation, ligament formation and fragmentation, at low particle density. . . . 84

4.12 Geyser simulation (left to right): purely ballistic particle system - top, our model with reduced surface tension (×0.1) - second from top, our model - second from bottom, our model with increased surface tension (×10) - bottom. Low surface tension interactions cause many satellites to be emitted and create a finer spray effect, whilst high surface tension tends to coalescence and causes system to tend to fewer, larger droplets. 85
4.13 Fountain-like jet (1.3m particles): Top - Droplet interactions create a varied droplet size distribution and therefore drag strength, which causes larger-scale break-up of the flow. Bottom - No interaction exposes artefacts in emitter and timestepping, these would usually require artist intervention and re-simulation to fix. 86
4.14 Times of collision detection and collision resolution (incl. outcome determination and satellite emission) operations for simulation in Figure 4.13. 480 frames with 2 substeps per frame, $\Delta t = 1/96$. 87

5.1 Results of a FLIP liquid simulation. a) Final surfaced result using our method. b) Particles coloured by their calculated value for k_s using Equation 5.7, values vary from 114—553 (droplets removed, color map: 144 - yellow to 553 - blue). Variation in this parameter occurs at the surface and at areas of interest where droplets break off from the main liquid body. 95
5.2 Comparison of surfaces with different ellipsoid normalisation values. The ‘local’ method (a) better preserves sheets at the edges whereas ‘global’ (b) can shrink the particle imprints at the edges to produce thinner sheets more prone to holes. As the values for this are dependent on the input particle set and other parameters of the model, i.e. search radius R, using the wrong value can lead to undesirable results as we see in c). 97
5.3 Alternative methods for particle surfacing. Both the bumpy surface of the isotropic distance function and the artefacts on the method of Solenthaler and Pajarola [2008] would require post-processing to create a suitable liquid surface. Even then these would fail to capture fine details such as thin sheets and droplets as well as the method we describe. 98
5.4 A whale splash simulation taken from DNEG’s training material, surfaced using the method described in this chapter. Note the small droplet details and smooth surfaces, especially in the splash. 99
5.5 Statistics for uses of the described Point Surfacer node in Houdini at DNEG, for 30 days prior to September 21st 2018. Graphs taken directly from Kibana. 100

6.1 The structure of this tool and how it integrates with a chosen application, broken down into the two main components, the compiler and the executable. 104
6.2 A particle simulation step using gravity and drag against a constant wind force. . . 107
6.3 Decimating points in a set using a percentage threshold. 107
6.5 Statistics for uses of the OpenVDB AX node in Clarisse at DNEG, for 30 days prior to September 21st 2018. 110

List of Tables

3.1 Table of Pressure Solve times with different pressure projection operators for 128^2 smoke plume simulation in Figure 3.1. 41
3.2 Table of times of a Preconditioned Conjugate Gradient Pressure solve using Proper Orthogonal Decomposition (POD) and combination preconditioners for initial pass and Incomplete Cholesky in CG loop. 49

4.1 Suggested values for the parameters considering material properties of water/droplets. The relative nature of the collision parameters makes these somewhat flexible and allows artistic control through their modification e.g. modelling the approximate behaviour of larger ‘droplets’ than those in reality (Figure 4.12) 77
4.2 Simulation times for results shown. The complexity of the operations, including the emission and deletion of particles, and widely varying particle count arising from interactions means results can differ considerably. All simulations performed with two 8-core 3.10Ghz Intel Xeon CPUs and 64GB RAM. 79

6.1 Performance of C++ implementations vs our JIT-compiled expression examples running on 32 core Intel Xeon 3.10Ghz CPU with 64GB RAM. 108

Acknowledgements

I would first like to thank my supervisor Richard Southern for his guidance, knowledge and experience, all of which have been key to helping me understand what it means to be a successful researcher. A huge thank you to everyone else at the Centre for Digital Entertainment past and present, especially Dan Cox, Zoe Leonard and Mike Board for their constant support and encouragement. I must also thank the EPSRC for the funding through grant EP/G037736/1, without which this research would not have been possible.

Alongside those above, my CDE colleagues deserve thanks for sharing their experiences and their struggles, helping me to find the sometimes difficult balance between research and industrial interests. A particular thank you to Alexandros Gouvatsos and Rahul Dey for their continued comradery and friendship over countless cups of coffee, plates of currywurst and bowls of ramen.

I would also like to thank those at DNEG that I have worked with and learned from over the years. Particular thanks to Nick Avramoussis, Matt Warner and the wider FX R&D team for their support with all things code and fluid based, and for their important contributions to my development and the successes of this project. Thanks to Ian Masters and Jeff Clifford for seeing the benefit of driving research via industrial experience and giving me the opportunity to come and work with a world-class VFX studio such as DNEG.

Thank you to my friends for their continued excitement and interest in my research, particularly Chris Brown who has allowed many an evening to descend into listening to me talk about water droplets and why they so are important.

My sincere thanks to my family for their unwavering support, forever encouraging me to 'find a job I love so I never have to work again'.

Finally, the biggest thanks to my best friend and partner Claire. Thanks for the walks through the park to the office on weekends; for the holidays booked as deadlines - and the motivation to reach them; for reading my thesis cover-to-cover and for the million other ways you continue to show your love and belief in me, always.

Author's Declaration

This thesis contains work from 2 publications:

- *Physically-Based Droplet Interaction*, 2017, R. Jones & R. Southern, Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2017.
- *A JIT Expression Language for Fast Manipulation of VDB Points and Volumes*, 2018, N. Avramoussis, R. Jones. F. Gochez, T. Keeler & M. Warner, Proceedings of DigiPro 2018.

The former being the sole work of the author of this thesis and the latter being a collaboration between the author and the FX R&D team at DNEG. The contribution of the author of this thesis to the latter consists of work towards the use of this language as a point simulation tool, for example the ability to move points in space.

Chapter 1

Introduction

1.1 Motivation

Liquid effects are a huge part of the modern visual effects (VFX) toolkit. With physical phenomena such as liquids difficult to both create and control in reality on a production set (especially beyond the scale of a wave-pool or splash), it is often up to visual effects artists to create natural and plausible looking CG versions of these phenomena for feature films, TV and animation. These recreations range from huge-scale tsunamis flooding city streets, to small-scale 'crown splashes' in glasses of milk, and everything in-between. Complex behaviours such as those of liquids would be extremely difficult and time-consuming for even the most skilled artist to hand animate, so the common approach (as with many other similar physical phenomena) is to create these effects using simulations. Liquid simulations in VFX track the shapes and dynamics of liquid using 3D CG geometry[1] which is then rendered through a virtual camera to create a final 2D effect that can be composited into a film. In order to recreate realistic liquid motion, these simulations are *physically-based*, that is, driven by the same physical processes that govern real-world liquid behaviours. However, compared to applications such as engineering, where physical accuracy is paramount, the only thing that ultimately matters from liquid simulations for VFX is that the results look plausible and achieve the desired visual impact. In this way, the methods used take advantage of a collection of simplifications to allow practical simulation times and control of many behaviours in the results. Even with these simplifications, liquid simulations (especially large-scale ones) commonly take hours, if not days, to simulate. The amount of data created from them can also be huge (an extreme, albeit not unprecedented, example: a single simulation on Disney's sea-based adventure *Moana* required over 20 TB to store [Palmer et al. 2017]).

As the use of visual effects in feature film and TV grows in search of more dramatic and awe-inspiring visuals, requirements on both scale and quality are similarly ever-increasing [Visual Effects Society 2013]. For simulated effects like liquids, this generally means increasing resolutions (i.e. number of discrete elements the simulation is composed of) or the consideration of more complex behaviours, both of which generally lead to more time being spent on setting up and running these simulations. This places demands on VFX studios to use more costly

[1]These can also be done in 2D, but most techniques in VFX use full 3D recreations.

resources such as computing power and artist time which are in huge demand from all other areas of production.

Workflows in VFX for creating simulated phenomena are largely-based around the same systems as those used by the wider computer graphics community. However, the requirement for effects in feature film to be as photoreal as possible can often expose shortcomings in the methods used elsewhere. Whilst liquid simulation and the creation of liquid effects are recognised as a key component of the effects tool-set by both industry and academia, some of the most important aspects of the creation of realistic (or at least, plausible) effects have been largely overlooked. In particular those of droplets, sprays and other visibly complex liquid phenomena. Most research focuses on simple free-surface liquids but it is often these other elements that are crucial in grounding liquid effects in reality. In lieu of adequate tooling for these components, their creation relies heavily on alternative approaches such as non-physical methods and appropriating other physical simulations, or through purely bespoke setups, manual labour and artistic ingenuity.

In theory, all of these components could be the by-product of having sufficiently high resolution and modelling all phases and interactions in a single simulation of liquid and air phenomena. However, in practice, *sufficient* resolution to capture all of these scales of motion (large: oceans, waves; to small: droplets, sprays) would be orders of magnitude higher than those used currently in production and the physical models would need to incorporate complex multi-phase dynamics. Such requirements would lead to unfeasible computational complexity/simulation times, much reduced artistic control and would be extremely difficult to implement and incorporate into the visual effects pipeline. As such, in the standard visual effects workflow, liquid effects are comprised of many layers of simulations composited together, often incorporating a collection of procedural or simplified reproductions of the elements that are not captured when using a practical resolution, single-phase, free-surface liquid simulation (e.g. particle simulations for droplets, sprays etc.).

The layered pipeline used does however have its benefits, it offers a lot of control to the artist in the creation of each separate element and allows elements to be created an approved by the director/visual effects supervisor independently. For example, the main body of a wave splash may be approved whilst the artist is able to make successive attempts at finer spray motion without requiring changes to the initial splash. This avoids expensive re-simulation and the trouble of having to make sure inter-dependent elements are perfect simultaneously.

However, this workflow still requires high quality methods for the actual creation of these other elements themselves. Within the graphics community, there has been a wealth of research on liquid simulation over the last decade or two, yet the tools used in production remain largely unchanged from a decade ago and in particular the methods for elements such as droplets and sprays have seen almost no development. As most research into fluid simulation looks at either smoke-like gas simulation or free-surface liquid flow, there do not exist many methods capable of creating these finer-scale liquid (often multi-phase) phenomena out-of-the-box. Instead artists are required to use procedural and simplified elements to create these effects - such as ballistic particle systems for droplets, sprays and foam or ill-suited physical simulations such as smoke simulations for mist and finer sprays.

Even once the simulations have been run, there remains the non-trivial task of creating final renderables from the simulated data. Given the variety in methods for creating these simulations, these post-processing tasks require similar amounts of care and attention. It is particularly difficult to find methods that are robust to the variations between different simulations and so, many of the workflows for these stages in the pipeline are also based on large amounts of artistic input.

To facilitate such artistically driven workflows, especially for simulated phenomena, is a technical challenge. Tools given to artists should be easy-to-use for as many scenarios as possible, but with the capability to allow the complex bespoke functionality that often ends up being required. Many applications used throughout VFX expose very high levels of control to allow these workflows, but knowledge of these tools varies between artists and so this practice can place high demands on more technical artists to perform lower-level modifications.

Considering all of these issues, the hypothesis behind this project is that an increased level-of-detail captured by the physically-based components of liquid simulations would improve these workflows. As many of these are heavily reliant on artistic input (and so, time) to reproduce all elements required for physical plausibility, improvements to these workflows would allow artists in VFX production to focus efforts on creative direction instead.

This thesis explores a few different approaches to achieve increased levels of detail in these simulations. The main approaches taken look to make these improvements through performance gains (via model reduction, Chapter 3) and direct simulation of finer-scale elements through droplet/spray modelling (Chapter 4). Also included in this thesis are developments that improve surface details coming out of existing simulations and tools for improved artistic control of simulations and their data (Chapters 5 & 6).

1.2 Academic Context

The creation of liquid phenomena in computer graphics has been studied for decades. The methods developed by graphics researchers and practitioners have been very successful at creating a variety of liquid effects such as the waves on oceans and splashes in water. From this, it has become commonplace for these methods (FLIP/PIC in particular, §2.3) to be used in the creation of liquid phenomena for visual effects. However, there are few methods that have been developed to handle the wide range of elements that occur in more energetic liquid phenomena such as splashes with sprays and droplets, and even fewer still that create photoreal results without large amounts of artistic intervention. In this thesis we look to build on the successes of previous researchers and develop new methods to help to ease the creation of photoreal liquid phenomena. We will further discuss the theory behind liquid simulation, the methods used in graphics, and the issues surrounding recreating splashing liquids in particular in Chapter 2.

1.3 Industrial Context

This project has been undertaken in partnership with a world-leading visual effects studio, DNEG. This has offered unique insight into the realities of production, which has helped to drive projects and focus our work to tackle real problems. Given DNEG's technological background

(§1.3.2), this has also given a great platform for work to build upon and has allowed seamless integration of the approaches developed into a real-world production pipeline.

1.3.1 VFX

VFX are a huge part of the modern creative process involved in the creation of feature films, and more recently, television. They act as as way of capturing visual imagery that would be impossible, dangerous or too expensive to capture in-camera [Visual Effects Society 2013]. Modern techniques make heavy use of computational tools for all elements of the pipeline from the initial tracking of the motion of actors captured on film, the creation of fully computer-generated effects like explosions or collapsing buildings, to the composition of the final frames that are seen on screen.

Effects in VFX

The VFX pipeline consists of many different departments and creative specialties. The stage in the pipeline that considers the creation of dynamic, non-character, often physically-based, phenomena such as fire, water, smoke and destruction is known as Effects (FX), with work performed by FX artists. These phenomena are generally created using simulations, often based-on or at least motivated-by the physical processes that occur in reality. Due to the ubiquity and the quantity of FX work required for feature films, alongside the difficulty of creating effects that are both physically-plausible and creatively suitable, this is a huge part of the work done by VFX studios. From a 2016 study by Barber et al. [2016] (Figure 1.1), we can see that FX work takes around 14% of the hands-on artist time of all the work done by a typical VFX studio (in this case, DNEG §1.3.2). Further still, we estimate that due to the comparatively computationally expensive nature of simulation, the cost of computational resources used by FX would likely be a higher proportion than this (both in hardware requirements and actual CPU time).

Liquids are one such natural phenomenon that are often required to be created by FX artists, the scale of which can range all the way from massive oceans to single droplets. Whilst it is not uncommon for any large summer blockbuster to require some form of liquid effects, there have also been plenty of films that would likely be impossible to make without them, such as *Life of Pi* [Lee 2012], *In the Heart of the Sea* [Howard 2015] or any of the *Pirates of the Caribbean* series (e.g. Ronning and Sandberg [2017]). Due to the complex nature of liquid motion and the range of phenomena this incorporates, these FX can require significant engineering by artists to create plausible recreations - this is the motivation for our work in this project.

1.3.2 Industrial Partner: DNEG

A VFX studio that has a great history of creating liquid FX is DNEG, the industrial partner of this research. DNEG (formally Double Negative) is a 4-time Oscar winning VFX studio. Formed in 1998 in London, DNEG now also has a collection of facilities in Vancouver, Montreal, Los Angeles, Mumbai and Chennai, employing over 5000 people worldwide. Recent achievements include Oscars and BAFTAs for their work on Blade Runner 2049 [Villeneuve 2017], Ex Machina [Garland 2014], Interstellar [Nolan 2014] and Inception [Nolan 2010]. Other notable recent

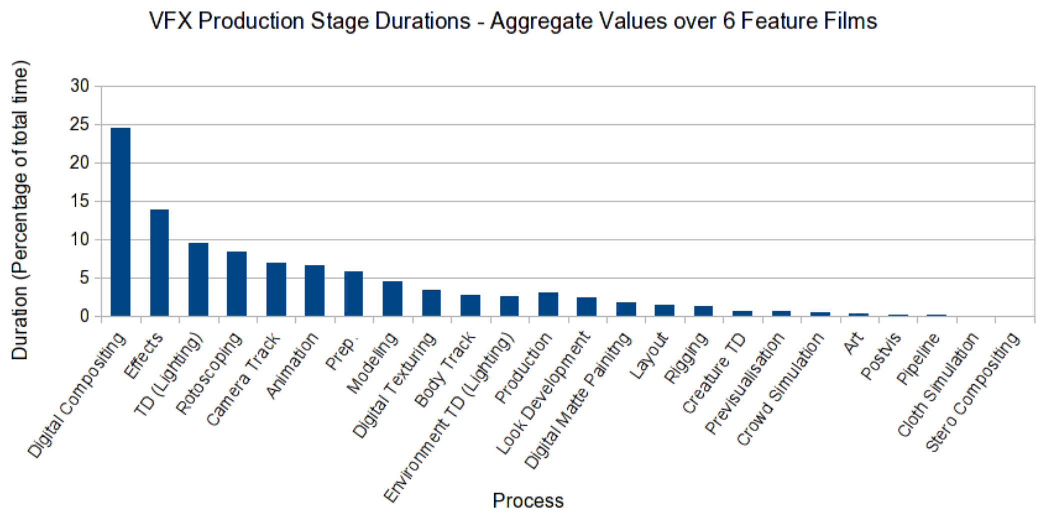


Figure 1.1: Time spent on different stages of production (hands-on artist time) at DNEG on 6 productions from 2014-2016. Effects development takes around 14% of the time, second only to compositing (which is required for every shot of every film). It is our understanding that this remains representative of the work undertaken in 2018. Figure taken from Barber et al. [2016].

works include Venom [Fleischer 2018], Pacific Rim: Uprising [DeKnight 2018] and Dunkirk [Nolan 2017]. DNEG also has a division working on VFX for TV and has recently branched out into feature animation in partnership with the UK's first high-end feature animation studio, Locksmith Animation.

Effects R&D at DNEG

As an industry that has often been the driving force for pushing development of the technology it uses, it is not uncommon for VFX studios to have in-house software development teams. In London alone, DNEG has a Research and Development (R&D) department of around 70 people working on a collection of both proprietary and open-source tools. The work undertaken by this department includes everything from developing tools to aid the transfer of data between sites and different applications, to development of proprietary simulation software. Of particular relevance to FX work at DNEG, is a state-of-the-art liquid solver, Dynamo, as well as ongoing contributions to OpenVDB, an efficient, industry-standard tool-set for sparse volumetric data.

Dynamo is a simulation framework developed at DNEG which allows FX simulations to be authored inside of Houdini [Side Effects 2018] (a popular Digital Content Creation program (DCC) for FX development) and run standalone on the render farm. These simulations can also be distributed amongst a collection of machines to increase simulation performance and avoid memory limits of a single machine. This incorporates a highly-efficient implementation of a hybrid particle-grid (FLIP/PIC) liquid simulator, the industry standard for liquid simulation (more on this in Chapter 2). In fact, this simulator is based on DNEG's original Squirt liquid solver, which is to our knowledge the first FLIP/PIC liquid solver to be developed for VFX production. Following recent developments this is able to handle huge-scale simulations, e.g. billions of particles and as such facilitates the extremely high resolution simulations that are in demand for high quality feature film VFX [Bailey et al. 2015].

OpenVDB [Museth et al. 2015] is an open-source C++ library for the representation and manipulation of volumetric data in sparse grids called VDBs. These are used for the storage of discrete, grid-based 3D data such as scalar and vector fields as well as Signed Distance Functions (SDFs). VDBs are a tree-based data structure, used to store spatially sparse data whilst supporting fast access, as well as insertion and deletion of values. Originally developed at Dreamworks Animation, this library and the VDB data structure has since been integrated into many of the widely used DCCs in VFX and animation and is the industry standard for volumetric data. When this project began, DNEG had just released an open-source extension for this library (originally developed for use in Dynamo) to handle point data. This project has since coincided with a large amount of the development of this extension library, known as OpenVDB Points, which is now included in the core OpenVDB distribution. The work undertaken in this project, when dealing with volumetric or point data been developed primarily with this format and has helped drive its development into a more comprehensive tool-set. In this way this project has both indirectly and directly (Chapter 6) contributed to the tools now widely available in this open-source library.

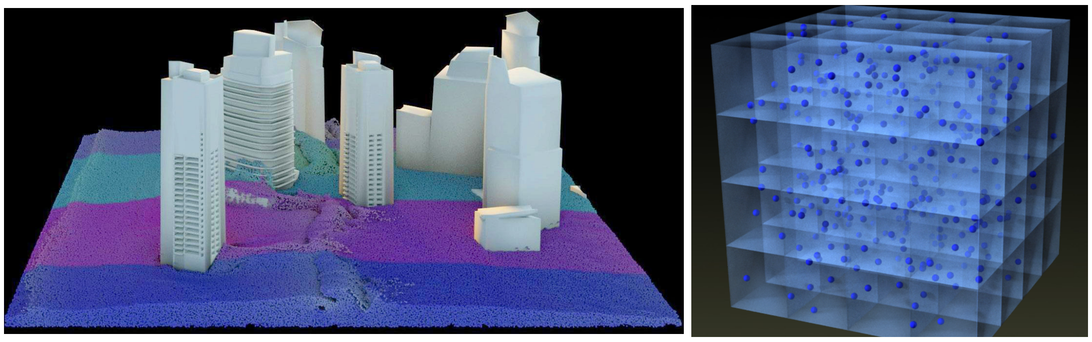


Figure 1.2: R&D at DNEG develops tools to solve a large range of issues in VFX production. These include the development of solvers capable to handling massive data-sets such as the distributed Dynamo Liquid solver (left - image taken from Bailey et al. [2015]) and highly efficient ways of storing and using geometry data such as OpenVDB Points (right - image taken from Museth et al. [2015]).

Original Industrial Partner: Prime Focus World

At the beginning of the research project the industrial partner was a different VFX studio, Prime Focus World. The project described in Chapter 3 was started during this time. Following the downsizing of their London office during the first year of this project, the partnership was severed and a new industrial partner was found in DNEG. Due to the different technology available at DNEG, and their greater experience in production VFX, the direction of the project changed to better focus on problems relevant to high-end production VFX. The remaining projects, and majority of work described in this thesis, are the product of the partnership with DNEG.

1.3.3 Considerations for Industry-Focused Research

An important characteristic of this project was the influence of the industrial context on its design and direction of decisions. All work has been developed using industry standard tools, techniques, and data structures, often open-source (i.e. OpenVDB, §1.3.2) to maximise the reproducibility elsewhere in production. Similarly the experiments and products of this project have been integrated into production workflows through use of software such as Houdini [Side Effects 2018]. As we will see later, this has also led to the development of more generally applicable production-focused tools, in particular the OpenVDB AX Compiler (Chapter 6).

Given the industrial focus and motivation, some key considerations have been made for applying novel graphics research within this context. Most importantly it should be recognised that the use of technologies and the systems in place in production can be quite different from the idealised scenarios often considered by works from outside of this environment. From this, an important requirement for our work was that it would be able to be both integrated into existing pipelines (and/or simulation systems) and robust to artistic inputs and manipulation.

The tools used in production are far more mature than compared to say 20 years ago and as such methods for working have been well established. Tools such as Houdini provide a huge amount of artistic freedom, and shortcomings in quality of results are largely understood to arise from the problem of how quickly an effect can be created rather than if it is possible. In theory any shot or effect could be completed with current tools due to the flexibility that is given to artists, even if it requires an almost hand-animated approach. In reality however, there are still practical considerations with the FX tool-set, but it is important to recognise that robustness to artistic intervention is a requirement and these practices should be expected with any approach developed for use in production. In this way, research in this area should look to improve upon these methods of working, encouraging artistic freedom whilst allowing the creation of final quality results faster than previously possible. In our case, this meant developing tools that create more realistic behaviours out-of-the-box; allowing increased levels of control over the behaviours exhibited by our simulations; and exposing new ways to interact with these simulations and their resulting data.

1.4 Key Challenges of VFX Production

Before we introduce the methods used to create liquid FX, and the specific problems that they demonstrate, we should first note some of the key challenges that persist throughout all areas of VFX. These help to explain some of the general themes of tool development in production. The realities of production place specific demands on the tools to be used by artists including **photorealism**, **efficiency** and **art-directability**:

- *Photorealism:* the entire VFX pipeline is based around making effects that are seamless and indistinguishable from reality (or for fantastical effects, at least believable). Every effect that gets composited into a shot must capture all of the behaviours and visible details that the audience would expect. With regards to physical phenomena such as liquids, this means recreating all of the behaviours and visual components that occur in reality. To achieve this, effects are often built up of many layers, of simulated and non-simulated components, to create a final result. Whilst many of these layers are created using well-defined systems, such as larger-scale liquid surfaces being created using free-surface simulations, others will be composed of more artistically driven components, particularly for finer-scale phenomena that occur such as droplets or sprays - commonly simulated using basic particle systems. It is often these other components, rather than the primary simulations (i.e. free-surface liquid), that require the largest artistic efforts (§2.5).

- *Efficiency:* resolutions used in production often dwarf the examples used in academic research as artists aim for higher fidelity and larger-scales. A typical FLIP/PIC liquid simulation may consider tens or hundreds of millions of particles, a figure which is always rising [Bailey et al. 2015]. This has prompted a range of developments from within the production context to allow existing systems to be able to facilitate these demands. These include use of efficient data storage and simulation formats such as OpenVDB [Museth 2013; Bailey et al. 2014], as well as other considerations such as distributed computing [Bailey et al. 2015; Lait 2016]. As simulations of such high resolutions become possible using these technologies, this places burden elsewhere in the pipeline to have to deal with the storage, transfer and handling of these huge amounts of data. As such, careful design of workflows and pipelines must also be considered when dealing with the resolutions required by high-end VFX productions [Palmer et al. 2017].

- *Art-directability:* the main difference between the use of simulation in engineering compared to VFX, aside from the physical accuracy of the methodologies used, is artistic motivation. Whilst a simulation in engineering may give results that inform some future decisions or test some theory, in VFX the simulation only exists as a creative tool and is authored to fit to an artist's vision. In this way, artistic control over the results is possibly the most important aspect of simulation. Some works from the graphics community have tried to offer animation-like controls, such as guiding simulations to keyframes of smoke [Fattal and Lischinski 2004; Hong and Kim 2004; Treuille et al. 2003] and liquids [McNamara et al. 2004; Shi and Yu 2005; Shin and Kim 2007]. However in practice, artistic control often instead deals with either less well-defined notions of control such as the visual style of the effect or the size of the features it displays, or the possibility of much more specific requirements such as manual control of simulated elements. To that end, the FX tool-set has matured over the last decade or two to the point where it is now possible to exhibit almost all of these kinds of control through applications such as Houdini. Artists are able to create and compose a huge variety of operations using all types of geometry to achieve almost whatever effect they are looking for. Effectively defining visual programming languages, specialised for geometric and simulated effects, these types of applications have been widely adopted throughout the industry and are hugely favoured by artists over previous, more black-box tools. Outside of Houdini, this also includes other DCCs such as Maya, Blender and 3DS MAX. In fact, other similarly flexible toolkits have also been developed by VFX studios themselves such as Rapid [Penney and Zafar 2015], Dataflow [Hankins et al. 2015] and Dynamo (§1.3.2) to offer even more control.

Even well-established simulation systems see a huge variety of modifications in production. Variations on these methods are created either by the tools developers themselves, e.g Houdini's native liquid solver offering the ability to allow particles in low density regions to break off from the main body, or by artists as needs arise e.g. deleting fast moving particles or adding arbitrary forces (§2.5). The tool-set allows artists a huge amount of freedom to access the data within the simulation and fix almost any issues in their results, or even develop entirely novel behaviours. However, that is not to say that all problems are solved by simply exposing control to an artist, instead it is that for any given situation or shot, an artist would have the ability to hand-tune the results, if required. Whilst this opens the door for a huge variety of effects and the ability to deal with plenty of issues, such a workflow is not feasible on a grand scale and should be reserved for only very specific artistic challenges.

We will look further at these challenges in the context of the production of high quality liquid FX in Chapter 2.

1.5 Project Objectives

The goal of this project is to improve the ability of artists to make high-quality liquid effects created in VFX production. To do this we will explore novel and improved methods for creating liquid simulations, as well as better ways of working with the resulting data. A key focus of this work, crucial for production impact, is the art-directability and efficiency of the methods used, as well as making sure that developments fit into the production pipeline and are able to compliment existing workflows.

1.6 Contributions

In this thesis we focus on increasing the levels-of-detail in recreations of liquid phenomena and improving workflows for artists in VFX to achieve production quality results. To reach this goal we have developed new and improved methods that incorporate more realistic behaviours and fine details within existing simulation frameworks as well as providing new tools for working with these systems, all of which have been designed such that they can be readily integrated into production workflows across the industry. To demonstrate this, some of these have already been deployed in the DNEG production pipeline and seen use on a collection of films such as *Blade Runner 2049* [Villeneuve 2017], *Pacific Rim: Uprising* [DeKnight 2018], *Venom* [Fleischer 2018], *The Kid Who Would Be King* [Cornish 2019] and *Godzilla: King of the Monsters* [Dougherty 2019].

The main contributions of this project can be summarised by the following:

- A review of the practical approaches used for creating highly-detailed effects of splashing liquids in VFX production (Chapter 2).
- An investigation into the use of model reduction for re-simulation and its suitability for application to hybrid simulation methods (Chapter 3).
- The development of a novel approach to handling realistic droplet interaction within particle systems for spray simulation, incorporating small-scale droplet dynamics to allow use for microscopic simulations as well as driving larger spray phenomena (Chapter 4).
- Improvements to a state-of-the-art particle surfacing algorithm to make it suitable for production use (Chapter 5).
- The development of an efficient and portable geometry manipulation expression language built on top of the industry standard for volumetric data, OpenVDB, allowing greater artistic control through user-defined operations on points and volumes (Chapter 6).

1.6.1 Publications

From these contributions, the following publications have been produced:

- *Physically-Based Droplet Interaction*, 2017, R. Jones & R. Southern, Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2017.
- *A JIT Expression Language for Fast Manipulation of VDB Points and Volumes*, 2018, N. Avramoussis, R. Jones. F. Gochez, T. Keeler & M. Warner, Proceedings of DigiPro 2018.

1.6.2 Software

- *OpenVDB AX* - Released by **DNEG** - N. Avramoussis, R. Jones. F. Gochez & M. Warner.

1.7 Thesis Outline

At the beginning of this thesis we shall set out the research context. Chapter 2 gives the mathematical and theoretical background to liquid simulation methods used in graphics before describing the workflow for creation of liquid effects in VFX and in particular, the efforts required to reach production quality results.

This is followed by chapters describing the research projects undertaken to improve this workflow in Chapters 3 - 6. To take advantage of the iterative nature of authoring liquid FX in Chapter 3 we discuss work towards improving efficiency using previous simulation data. Chapter 4 then documents a project on introducing plausible fine-scale droplet behaviours into widely used ballistic particle systems, increasing the levels of detail able to be captured by these systems. Following this, Chapter 5 describes the integration of a state-of-the-art particle surfacing method into the production pipeline and the required extensions for this model to make it robust to the inputs that may arise during use in production. The last project, discussed in Chapter 6, focuses on the work undertaken towards development of a new programming interface for artistic control of particle and volume data, used in both FX simulation and elsewhere throughout the pipeline.

Finally, Chapter 7 will include discussion of this project as a whole, areas for future work and conclusions that can be drawn from this research.

Chapter 2

Liquid Simulation for VFX

In this chapter we examine the theoretical background behind fluid motion and the approaches used to simulate this for computer graphics. We then explore how this translates into production techniques used in VFX.

2.1 Fundamentals of Fluid Motion

No thesis on liquid simulation would be complete without an introduction to the processes behind fluid motion, and in particular the way we are able to formulate these mathematically, through the Navier-Stokes equations.

The Navier-Stokes Equations describe the motion of fluids as the following set of partial differential equations:

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{g} \qquad \textbf{Navier-Stokes Equations} \qquad (2.1)$$

for $\mathbf{u} \in \Omega$, where Ω is the fluid domain, $\mathbf{u}$ the fluid velocity, $\frac{D\mathbf{u}}{Dt}$ is the material derivative of velocity[1], ρ the density, p is the pressure, μ the coefficient of kinematic viscosity and $\mathbf{g}$ is the external force, usually simply gravity. These describe the processes behind liquid motion in terms of the acceleration of the fluid at every point in the domain, and can be derived from mass and momentum conservation. They are composed of a convective term, $\frac{D\mathbf{u}}{Dt}$; a pressure term, $-\frac{1}{\rho}\nabla p$; a viscous dissipation term, $\mu\nabla^2\mathbf{u}$ and an external force term i.e. gravity $\mathbf{g}$.

If we further assume that fluid is incompressible i.e. cannot change material density, then we have the following additional constraint:

$$\nabla \cdot \mathbf{u} = 0 \qquad (2.2)$$

Whilst viscosity can be an important behaviour to incorporate for certain fluids such as honey or treacle, for largely inviscid fluids (such as smoke and water) we can simplify these

[1]This is the full derivative of the velocity $\mathbf{u}(\mathbf{x}, t)$ w.r.t. time, by the chain rule we have $\frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}$ for $\mathbf{u}$ in the Eulerian frame.

(taking $\mu = 0$) into the incompressible Euler equations:

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \mathbf{g} \qquad \textbf{Incompressible Euler Equations} \tag{2.3a}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.3b}$$

Solutions to the Incompressible Euler Equations

A common practice used in computer graphics to solve this complex system of PDEs is to use the concept of operator splitting [Chorin 1968] (introduced to this context by [Stam 1999]). That is, splitting the above equation(s) into a collection of smaller, simpler systems which can then be solved separately in sequence. Using this methodology, Equation 2.3 becomes a collection of force, pressure projection and advection steps:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{g} \qquad \textbf{Force} \tag{2.4}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0 \qquad \textbf{Pressure Projection} \tag{2.5a}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.5b}$$

$$\frac{D\mathbf{u}}{Dt} = 0 \qquad \textbf{Advection} \tag{2.6}$$

Splitting the Euler equations in this way allows the use of efficient methods to solve each separate component of the system, allowing flexibility and variety in the development of simulation frameworks which we will see later.

Solving these three components in sequence, using the solution to the previous stage as the input for the next, gives the required solution to Equation 2.3. However, it should be noted that the order of these operations is important - to preserve the incompressibility constraint, advection should be performed on a divergence-free velocity field. Therefore, a normal timestep (from t to $t + \Delta t$) in a fluid simulation can be described as follows:

$$\mathbf{u}^t \overset{\textit{forces}}{\rightarrow} \tilde{\mathbf{u}}^t \overset{\textit{project}}{\rightarrow} \hat{\mathbf{u}}^t \overset{\textit{advect}}{\rightarrow} \mathbf{u}^{t+\Delta t} \tag{2.7}$$

Given the solution to the velocity problem Equation 2.3, it is then possible to update the fluid state using this velocity, i.e. move the position of the surface of a liquid or track the evolution of smoke concentrations through time. This sequence of operations will evolve a fluid system through a single timestep, and so by repeating this process, create a fluid simulation.

We will now look at the various ways that these equations have been solved in applications to computer graphics, and the particular methods favoured in VFX. For more information on approaches to fluid simulation used in other fields such as engineering see Anderson and Wendt [1995].

2.2 Fluid Simulation for Computer Graphics

Fluid phenomena required in computer graphics come in a huge variety of forms including smoke, fire, oceans, waves, splashes and droplets, all of whose motion are driven by some forms of Equations 2.1 or 2.3. Such a wide variety of phenomena leads further still to an even wider range of approaches taken. We will focus on the theoretical contributions that act as the basis for the methods widely used in high-end VFX production specifically for creating highly-detailed, splashing liquid effects.

In general, there are three main types of methods used in liquid simulation for computer graphics, characterised by their different discretisations of the fluid volume[2]. These are: grid-based Eulerian simulations (Figure 2.1 - left), particle-based Lagrangian simulations (Figure 2.1 - middle) and hybrid simulations that use a combination of grids and particles (Figure 2.1 - right).

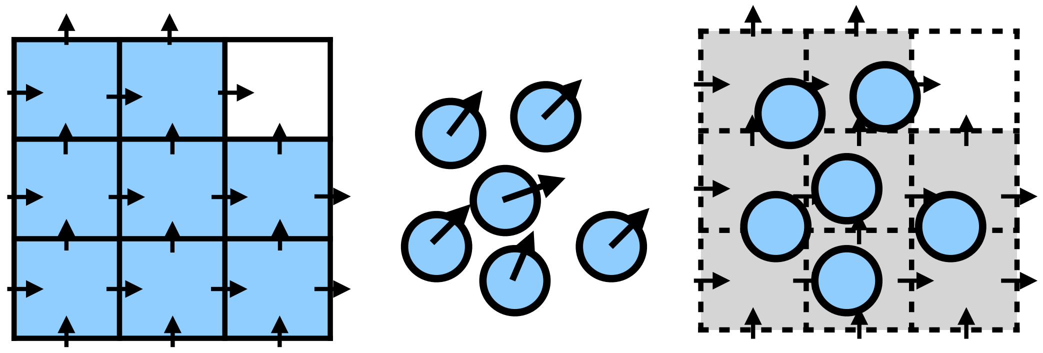


Figure 2.1: The three main discretisations used in liquid simulation for computer graphics. Left to right - **Eulerian**, **Lagrangian** and **Hybrid**.

Following the operator splitting approach, it has been found that some discretisations are more suited to solving particular stages of Equation 2.3 than others. For example, Eulerian methods, especially those using regular grids, can be very successful at solving the pressure projection step, Equation 2.5, as the problem can be formulated into a sparse linear system. Similarly, Lagrangian methods are very good at tracking motion and solving the advection problem, Equation 2.6, as they introduce very little numerical dissipation.

We will now discuss some of the seminal works using these methods and then consider the state-of-the-art before moving on to production-favoured approaches.

2.2.1 Eulerian Methods

Eulerian methods discretise the space of the domain in which we are interested, and track the fluid quantities at these discrete points over time. In this way each cell (discrete element in the domain)

[2]Other approaches not on this list include mesh-based and Lattice-Boltzmann methods, as well as other more bespoke and unique approaches.

will have values for velocity and/or density, temperature, fill volume etc. which describe the state of the fluid at any point in time.

Foster and Metaxas [1996] was the first work in graphics to consider full 3D liquid simulation, with prior methods only modelling reduced approximations to fluid motion such as the wave equation [Kass and Miller 1990]. Using a finite-difference approximation to the entirety of Equation 2.1 they are able to create liquid simulations by tracing marker particles through a regular grid on which the solutions are calculated. The Successive-Over-Relaxation solver used for the pressure update and finite-difference advection equation required many iterations and very small timesteps for stability. Following this, the now prolific operator splitting technique was introduced into graphics by Stam [1999]. In combination with their novel Semi-Lagrangian advection scheme - a method for tracing grid values backwards in time to solve the advection problem, Equation 2.6, this allowed much larger timesteps to be used. The solution to Equation 2.5 is formulated as a Poisson problem and through discretising with finite-differences, becomes that of solving a large sparse linear system. Foster and Fedkiw [2001] extend this approach to liquids tracking both marker particles and a implicit surface for the liquid interface. Also, recognising the discrete Poisson problem as a symmetric positive definite linear system, they introduce the use of Incomplete Cholesky Preconditioned Conjugate Gradient (ICPCG) solvers to solve this problem more quickly and efficiently.

Whilst liquid surface level-sets such as those used by Foster and Fedkiw [2001] allow partially full cells in the visual representation, the Poisson problem for Equation 2.5 (as in Stam [1999]) assumes either full or empty cells for the actual liquid itself, creating grid artefacts at boundaries. This was improved by Batty et al. [2007] who, by taking a variational approach to the pressure projection problem and considering the volume of fluid across the boundaries, allow handling of partially full and curved boundaries at solid-liquid interfaces. Ng et al. [2009] further extend this approach to second-order accuracy, instead considering the obstructed area across the cell face to determine fluid flow around solids. Similarly, for increased accuracy at the air-liquid interface, Gibou et al. [2002] developed the Ghost Fluid Method (GFM), using the liquid surface level-set values to set correct pressures at a sub-cell liquid interface rather than at the grid resolution as done previously.

As scales of simulations have increased, there have been a collection of works that look to use Multigrid (MG) methods to speed up the solution to the Poisson problem (or its variants) as it is often the most computationally expensive stage. McAdams et al. [2010] developed a MG preconditioner for the Preconditioned Conjugate Gradient (PCG) solve of a liquid simulation (albeit using using the original voxelised linear system [Stam 1999]) but they were unable to create a fully MG method that did not diverge or stagnate. Chentanez and Müller [2011] found that with the addition of variational solid boundaries [Batty et al. 2007] *and* GFM [Gibou et al. 2002], it was possible to use a fully MG solve for the pressure projection without these issues in the case of collocated pressure and velocity samples (rather than the usual staggered method, which we will see later §2.3). Weber et al. [2015] extend this work to use the usual staggered grid, in turn increasing convergence of the MG solver.

Even with the use of MG solvers, the pressure projection remains a large component of the cost of a fluid simulation. This has prompted the use of adaptive methods to further reduce the scale of the linear system to be solved. Adaptivity allows simulation of areas of interest at higher resolution and fidelity than those of low interest, such as deep under the fluid surface. This reduces the size of the linear system to be solved and so speeds up solution. Methods of introducing adaptivity include: use of an octree data structure in Losasso et al. [2004]; uniform grid coarsening in Lentine et al. [2010]; using tall cells deep under the surface in Chentanez and Müller [2011]; and a tetrahedral mesh in Ando et al. [2013]. However, adaptive methods require irregular data structures and can have some issues at the boundaries of different scales, inhibiting their use in practice.

2.2.2 Lagrangian Methods

Lagrangian methods discretise the fluid itself into particles and track the positions and velocities of these over time. In this way each particle represents some part of the fluid and the positions and attributes of the particles describe the state of the fluid at any time.

Müller et al. [2003] introduced the SPH framework of Monaghan [1994] to graphics applications as a liquid simulation framework that allowed interactive performance. This method uses radial kernels around each particle - looking into its neighbouring particles to approximate quantities of the fluid at particle positions. Often quantities such as density are used to enforce the fluid behaviours, in particular the pressure projection defined by the Equation 2.5. This has since become a very popular method and used for many applications as well as incorporating many different behaviors such as viscosity [Peer et al. 2015], improved turbulence [Bender et al. 2017], and multiple-fluid interactions [Yang et al. 2015]. For standard liquid simulation, there have been a wide variety of variations to this system, say those enforcing constant density explicitly [Müller et al. 2003], or instead non-divergence of velocity [Ihmsen et al. 2014a], or both [Bender and Koschier 2017]; as well as adaptive methods to speed up calculation [Adams et al. 2007; Solenthaler and Gross 2011; Winchenbach et al. 2017] for larger-scale scenarios. Other similar methods have also been developed such as Position-Based Fluids (PBF) by Macklin and Müller [2013].

Whilst the particles are able to intuitively carry some fluid quantities such as mass and velocity, the arbitrary nature of their positions (and so the samples for neighbouring particles) can make it difficult to calculate some quantities (especially derivatives). Similarly in areas of low particle density and at boundaries, issues with unbalanced calculations such as density can create artefacts such as the tensile instability. However, these methods are extremely popular with researchers and many works over recent years have been successful in tackling some of these issues. For a more detailed look at SPH methods we refer the reader to the review by Ihmsen [2014].

2.2.3 Hybrid Methods

Hybrid methods combine both Eulerian and Lagrangian approaches, using the most successful components of each.

In §2.2.1 we mentioned a method using both particles and grids [Foster and Fedkiw 2001]. Whilst the simulation was on the grid, particles were used to carry quantities and define the surface avoiding the issues of grid-based advection schemes. This is the first instance of hybrid methods used for liquid simulation in graphics, and has since become an extremely popular approach. Enright et al. [2002] extend this approach to the Particle Level Set (PLS) method, seeding particles around both sides of the liquid-air interface to track the surface more accurately. Zhu and Bridson [2005] introduced the FLIP/PIC method, a particle-grid method combining two different transfer methods between the discretisations, FLIP from Brackbill et al. [1988] with PIC from Harlow and Welch [1965]. This method uses particles throughout the liquid to perform the advection of fluid quantities, whilst using a background Eulerian grid (that gets populated by the values carried by the particles) to calculate the pressure projection and forces. The Langragian component avoids the numerical dissipation of Eulerian advection and the Eulerian component similarly avoids troublesome Lagrangian pressure calculations on the particles. Purely FLIP and PIC methods were previously too noisy and dissipative respectively for practical use, but their combination made for stable, yet energetic results.

Whilst a large amount of numerical dissipation is avoided in this approach (e.g. compared to that of Semi-Lagrangian advection), the transfer of quantities between particles and grids has been found to still introduce its own numerical issues, notably the dampening of angular momentum. The use of the combined FLIP and PIC transfer schemes somewhat alleviates the dampening of this transfer (their combination being what made this method feasible [Zhu and Bridson 2005]) but is equivalent to introducing high-frequency noise to the system. Jiang et al. [2015] recently introduced the Affine Particle-in-Cell (APIC) method to solve this issue, preserving angular momentum throughout transfer between particles and grids.

Further extensions to these approaches have looked to recapture some of the efficiency of approaches like PLS, such as that of Sato et al. [2018b]; or consider adaptivity such as Ando et al. [2012].

2.2.4 Summary

This review covers a relatively small collection of the work done in this field but most of the large developments that have been of adopted in VFX and other practical uses of computer graphics in the last 20 years. For the creation of smoke, gas and fire simulations, typically Eulerian methods are favoured. Whilst for liquids, particle-based (or hybrid) methods are instead used as they are able to better maintain and represent the liquid-air interface. Purely Lagrangian methods have seen a huge amount of research over recent years but for the scales of simulation used in VFX, demonstrating a good balance of efficiency and quality [Um et al. 2017], and perhaps most importantly due to the maturity of the workflows built around them, hybrid methods are the most commonly used in production VFX.

As these methods and their related workflows are the basis for the creation of the majority of liquid effects in production (and an important factor in decisions made later in this thesis), we will now give a detailed description of a hybrid liquid simulation framework.

2.3 FLIP/PIC Simulation

Possibly the most widely used method for liquid simulation in large-scale production VFX, is that of the particle-grid simulation FLIP/PIC [Zhu and Bridson 2005] (as well as its more

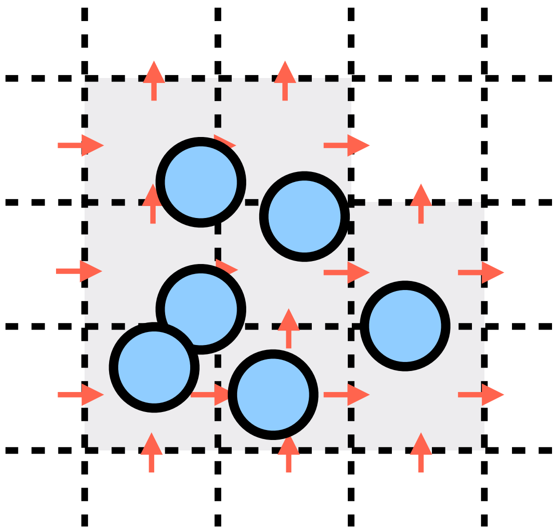


Figure 2.2: Standard particle and grid setup in FLIP/PIC/APIC simulations.

recent variant APIC [Jiang et al. 2015]). Both the implementation and artistic use of these methods are widely documented and they are available in many DCCs [Autodesk 2018; Side Effects 2018; Next Limit 2018; Blender 2018], as well as many VFX studios having proprietary implementations, e.g. DNEG's Dynamo.

As well as the computational benefits of this method (discussed in the previous section), there are workflow implications that have made it so successful. Firstly, particle systems are a very familiar concept to FX artists [Visual Effects Society 2013; Penney and Zafar 2015] 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 VFX artists.

As this is the standard used in production, to properly understand the how to work with and build upon this method (which we will do in future chapters), we will now describe the stages of a FLIP/PIC liquid simulation in detail.

A note on the background grid

For the grid representation of our fluid volume a staggered Marker-and-Cell (MAC) grid (Figure 2.3) is used, introduced by Harlow and Welch [1965]. This spatial discretisation allows the calculation of second-order accurate central differences at the exact positions that we require each quantity to be sampled [Bridson 2008]. By storing the velocity components on their corresponding cell faces and pressure values at the centre of the cells, we can intuitively

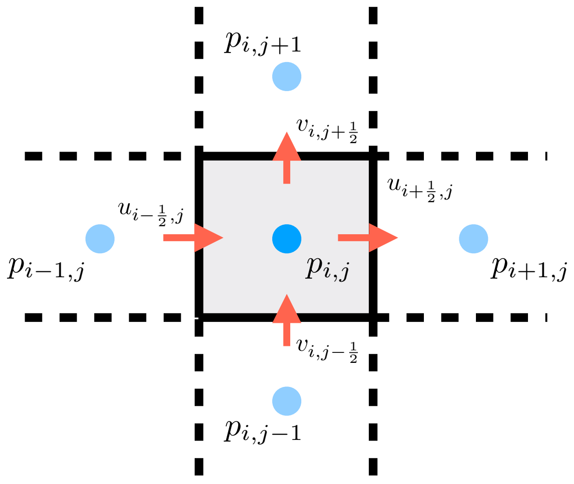


Figure 2.3: A 2-dimensional staggered MAC grid cell.

measure the divergence of velocity at the positions of the pressure samples, and the pressure gradients at the positions of the velocity samples. At the centre of grid cell (i,j,k) of width Δx we have:

$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} \tag{2.8}$$

and thus the divergence calculation becomes

$$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \approx \frac{u_{i+1/2,j,k} - u_{i-1/2,j,k}}{\Delta x} + \frac{v_{i,j+1/2,k} - v_{i,j-1/2,k}}{\Delta x} + \frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta x} \tag{2.9}$$

Furthermore, at the centre of the face (i—1/2,j,k) we have:

$$\frac{\partial p}{\partial x} \approx \frac{p_{i,j,k} - p_{i-1,j,k}}{\Delta x} \tag{2.10}$$

These avoids the need to interpolate values later, i.e. in the pressure projection.

2.3.1 Algorithm Overview

A typical FLIP/PIC fluid simulation will follow an algorithm such as that given in Algorithm 1. A user will typically supply an initial point set or geometry within which liquid particles will be emitted. The particles in the simulation will then undergo various operations and be moved around in 3D space following Algorithm 1. [3] The result of a simulation will most likely be the simulation particle set itself or a surface that has been created (or tracked) using it. This will then be post-processed and rendered to create a final free-surface liquid effect (more on this in Chapter 2.4).

Algorithm 1

```
1:  procedure FLIP/PIC FLUID SIMULATION
2:      input initial points $P_0$ with positions $\mathbf{x}_p^0$ and velocities $\mathbf{u}_p^0$
3:      for each timestep $t$ do
4:          for all grid cells $(i, j, k)$ do                          ▷ transfer to grid
5:              $\mathbf{u}_{ijk}^t = \sum_{p \in P_t} w_{(ijk)p} \mathbf{u}_p^t$ (Equation 2.11)
6:          end for
7:          for all grid cells $(i, j, k)$ do                          ▷ add forces
8:              $\tilde{\mathbf{u}}_{ijk}^t = \mathbf{u}_{ijk}^t + \Delta t \mathbf{g}_{ijk}$ (Equation 2.12)
9:          end for
10:         for all grid cells $(i, j, k)$ do                          ▷ perform pressure projection
11:             $\mathbf{b}_{ijk} = \nabla \cdot \tilde{\mathbf{u}}_{ijk}^t$ (Equation 2.9)
12:         end for
13:         solve $\mathbf{Ap} = \mathbf{b}$ (Equation 2.18)
14:         for all grid cells $(i, j, k)$ do
15:             $\hat{\mathbf{u}}_{ijk}^t = \tilde{\mathbf{u}}_{ijk}^t - \nabla \mathbf{p}_{ijk}$ (Equation 2.10)
16:         end for
17:         for all particles $p \in P_t$ do                            ▷ transfer from grid
18:             $\mathbf{u}_p^{t+1} = \text{sample}(\hat{\mathbf{u}}^t, \mathbf{x}_p^t)$ (Equation 2.20)
19:         end for
20:         for all particles $p \in P_t$ do                            ▷ advect particles
21:             $\mathbf{x}_p^{t+1} = \mathbf{x}_p^t + \Delta t \sum_i w_{ip} \hat{\mathbf{u}}_i^t$ (Equation 2.21)
22:         end for
23:         output points $P_t$
24:     end for
25: end procedure
```

Particle-to-grid transfer

The first step is to transfer the particle velocities onto the background grid. This is usually done with a weighted interpolation (e.g. trilinear) of surrounding particle values onto the velocity sample positions at the centre of each voxel face. For each component of velocity, say u, at its

[3] As well as main stages outlined here, there may also be stages where a liquid surface is calculated from the particles and/or particles may be re-seeded within the liquid volume, as well as stages where particles may break off from the main body and are dealt with separately from the main fluid, but we will stick to the description of the core algorithm for brevity.

corresponding position on the voxel face, $\mathbf{x}_i$ (as in Figure 2.3), we calculate:

$$u_i^t = \frac{\sum_p w_{ip} u_p^t}{\sum_p w_{ip}} \tag{2.11}$$

where u_i^t is the component grid velocity at $\mathbf{x}_i$, u_p^t is the component of particle velocity, and w_{ip} is the interpolant between the centre of face i at $\mathbf{x}_i$ and the particle at $\mathbf{x}_p$.

Applying forces

Now we have the velocity grid representing the fluid, say $\mathbf{u}^t = (u^t, v^t, w^t)$, we can update this with the acceleration due to gravity and other force inputs (Equation 2.4). As the external forces applied are usually considered constant for each timestep of length Δt, we can simply integrate this step with Forward Euler as follows:

$$\tilde{\mathbf{u}}^t = \mathbf{u}^t + \Delta t \mathbf{g} \tag{2.12}$$

Pressure projection

This updated velocity grid may now contain non-zero divergence values, causing the fluid to lose volume during advection. Therefore we perform the pressure projection step (Equation 2.5) to make the velocity field divergence-free and enforce boundary conditions such as collisions with solid geometry or the air-liquid interface.

The most common formulation of this system for computer graphics follows Chorin [1968]. That is, using the Helmholtz Hodge decomposition, any vector field $\tilde{\mathbf{u}}$ can be decomposed such that

$$\tilde{\mathbf{u}} = \hat{\mathbf{u}} + \nabla p \tag{2.13}$$

where $\hat{\mathbf{u}}$ is a divergence-free vector field s.t. $\hat{\mathbf{u}} \cdot \mathbf{n} = 0$ on the boundary of the domain and p is a scalar field. Therefore if we have p, we can define the orthogonal projection operator $\mathbb{P}$ that maps our velocity onto its divergence-free component by

$$\hat{\mathbf{u}} = \mathbb{P}\tilde{\mathbf{u}} = \tilde{\mathbf{u}} - \nabla p \tag{2.14}$$

To find p, Chorin [1968] show that, using Equation 2.13 and taking the divergence of both sides, we arrive at the Poisson problem:

$$-\frac{\Delta t}{\rho}\nabla^2 p = -\nabla \cdot \tilde{\mathbf{u}}^t \tag{2.15}$$

which will either have boundary condition:

$$\hat{\mathbf{u}}^t \cdot \mathbf{n} = \mathbf{u}_{solid}^t \cdot \mathbf{n} \tag{2.16}$$

where $\mathbf{u}_{solid}^t$ is the solid boundary velocity and $\mathbf{n}$ the outward facing normal of the solid boundary, or:

$$p = 0 \tag{2.17}$$

at the air-liquid boundary.

With our chosen discretisation of a staggered MAC grid, if we substitute the centred finite differences above into Equation 2.15, our system for p becomes a large sparse linear system (see [Bridson 2008] for details):

$$\mathbf{A}\mathbf{p} = \mathbf{b} \tag{2.18}$$

where $\mathbf{A}$ is a 7-point Laplacian matrix, $\mathbf{p}$ is our solution vector of pressures and $-\mathbf{b}$ is the vector of divergence values at each cell. In fluid simulations, this system is generally extremely large, with $\mathbf{A} \in \mathbb{R}^{N\times N}$ for N grid cells in our fluid domain. This is solved this using either a PCG or MG method as described in §2.2.1[4].

Given the solution pressure grid from this linear solve, we use the projection operator $\mathbb{P}$ on our velocity field by taking the gradient at each component velocity sample using Equation 2.10 and subtracting this from the velocity, making it divergence-free:

$$\hat{\mathbf{u}}^t = \tilde{\mathbf{u}}^t - \nabla p \tag{2.19}$$

Grid-to-particle transfer

This divergence-free velocity $\hat{\mathbf{u}}^t$ is then transferred back onto the particles. The FLIP/PIC method updates velocities on the particles with contributions from both a FLIP [Brackbill et al. 1988] and a PIC [Harlow and Welch 1965] sample of the fluid velocity grid at the particle position.

The FLIP sample consists of an *update* to the existing velocity value, adding the previous particle velocity and the interpolated difference between the previous grid velocity and updated grid velocity.

$$\mathbf{u}^{t+1}_{FLIP} = \mathbf{u}^t_p + \sum_i w_{ip}(\hat{\mathbf{u}}^t_i - \mathbf{u}^t_i)$$

where $\mathbf{u}^{t+1}_{FLIP}$ is our FLIP particle velocity, $\hat{\mathbf{u}}^t_i$ is the post-projection grid velocity, $\mathbf{u}^t_p$ is the previous particle velocity, $\mathbf{u}^t_i$ is the previous grid velocity, i are the surrounding grid values and w_{ip} is the trilinear interpolant between the face at $\mathbf{x}_i$ and the particle at $\mathbf{x}_p$.

The PIC sample is a simple interpolation of the updated grid velocity.

$$\mathbf{u}^{t+1}_{PIC} = \sum_i w_{ip}(\hat{\mathbf{u}}^t_i)$$

where $\mathbf{u}^{t+1}_{PIC}$ is our PIC particle velocity, $\hat{\mathbf{u}}^t_i$ is the post-projection grid velocity, i are the surrounding grid values and w_{ip} is the trilinear interpolant between the face at $\mathbf{x}_i$ and the particle at $\mathbf{x}_p$.

Our updated particle velocity then becomes:

$$\mathbf{u}^{t+1}_p = (1-\alpha)\mathbf{u}^{t+1}_{FLIP} + \alpha\mathbf{u}^{t+1}_{PIC} \tag{2.20}$$

where $0 \le \alpha \le 1$ is a chosen weight between FLIP and PIC samples.

[4]Even using state-of-the-art methods, this stage in the simulation quickly becomes a bottleneck as simulation size increases. This is a key motivation for our re-simulation work described in Chapter 3.

Advection

Hybrid fluid simulations use particle advection through the divergence-free velocity grid $\hat{\mathbf{u}}^t$. We take an interpolation of the grid velocity at the particle position (as in the PIC sample) and using this update our particle positions using our chosen integration scheme e.g. Forward Euler, RK2 or RK3. For example for a simple Forward Euler advection step we update each particle position $\mathbf{x}_p^t$ by:

$$\mathbf{x}_p^{t+1} = \mathbf{x}_p^t + \Delta t(\sum_i w_{ip}(\hat{\mathbf{u}}_i^t)) \tag{2.21}$$

The advection is the final stage of the simulation loop, after this the points are cached and the loop restarts for the next simulation step.

Key Characteristics of FLIP/PIC

As this is the standard method used in VFX production, there are some important characteristics to note that will affect later decisions made in this thesis:

- Whilst simulations are comparatively fast, they are dominated by the time to perform the pressure projection (Chapter 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 (Chapter 4).
- As the particles do not interact with each other directly, they are not guaranteed to be well-distributed which can create problems in later post-processes (Chapter 5).
- The simulations require manipulation of both point and volumetric data, and so artists may require ways to interact with both of these discretisations (Chapter 6).

2.4 Realistic Splashing Liquid Effects

Given the requirement for photorealism in feature film VFX, some particularly troublesome effects to create are those of liquid splashes and sprays, arguably some of the most visually exciting and dynamic physical phenomena. These occur in a variety of situations such as rapids, waterfalls and breaking waves - all scenarios which could be required for visual effects artists to recreate. As we have discussed in the previous chapter, the problem of making photoreal recreations of these kind of phenomena comes from the levels of detail required, and the variation in behaviours between each visual component.

2.4.1 Anatomy of a Splash

Consider the example of a splash shown in Figure 2.4 (left) and the annotated version Figure 2.5. Within this image of a breaking wave there are a large number of visible fluid components,

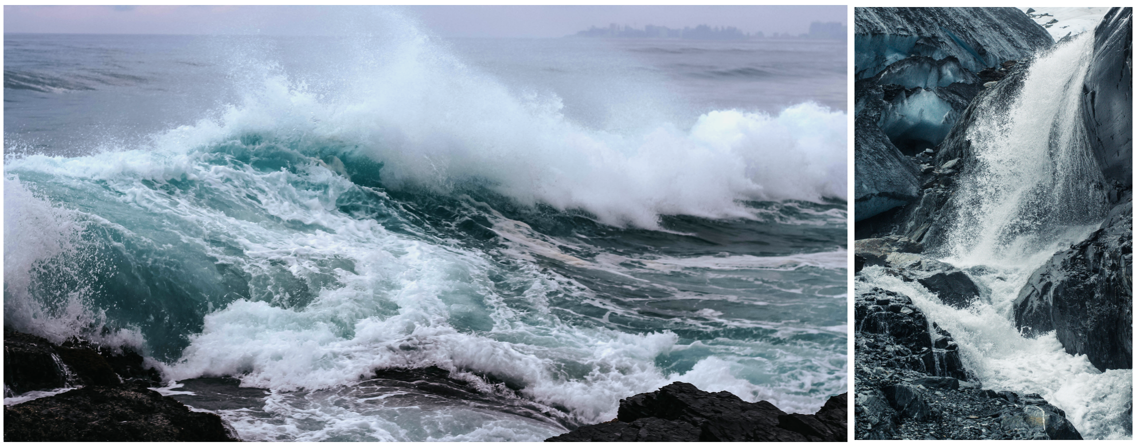


Figure 2.4: A wave breaking on the rocks (left) and a waterfall (right). Phenomena such as these are extremely complex, containing interactions between many different states of the liquid and air mixtures including droplets, sprays, foam, and bubbles, as well as larger scale liquid behaviours such as the wave breaking and the ocean wave dynamics.

all interacting in various ways to make up an extremely complex liquid phenomenon. These include:

- Ocean waves in the background;
- The main wave in the foreground, breaking over on itself;
- Fine sea spray tearing away from the peak of the wave;
- Droplets and splashes on the peak of the wave and against the rocks;
- Foam and churn in front of the wave near the shore;
- Bubbles entrained in the liquid creating lighter areas beneath the surface.

Also notice how most of the visibly dominant and arguably interesting components of this image are the droplets, foam and sprays. The larger ocean behaviour and wave breaking simply give a platform upon which these other components inform the viewer that they are looking at a violent, energetic liquid phenomenon.

Beyond Free-Surface Flow

Hypothetically, we could endeavour to simulate systems with high enough resolution and enough model complexity that we capture these small-scale features as well as larger motions in a single simulation, but such a mammoth task would be unfeasible in reality. In order to recreate all of these scales of motion at once with a single system would require impossibly high resolutions and a model incorporating a huge range of influences and interactions, incurring massive computational cost and even then, such a model would likely be extremely unwieldy

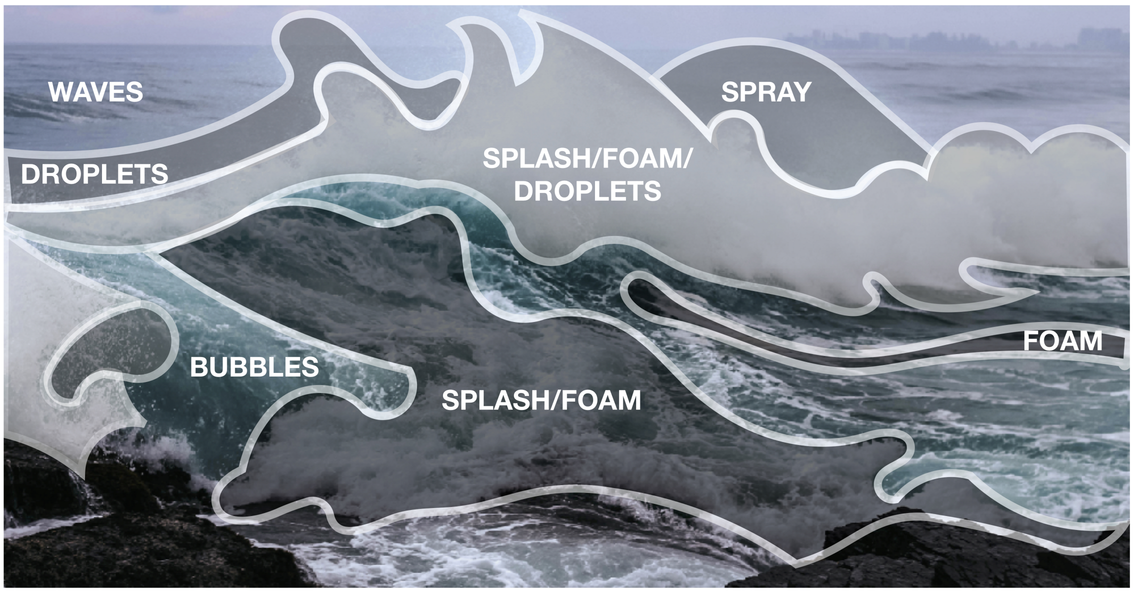


Figure 2.5: Dissecting the breaking wave from Figure 2.4 into its visible fluid components. Highlighting areas of predominantly splashes, foam, droplets and sprays. Note how the shape of the wave breaking and other similar larger-scale fluid behaviours are obscured by the smaller-scale fluid components.

as an artistic tool. Whilst work continues to progress on adaptive techniques (able to capture multiple scales of motion, spatially increasing resolution where required [Ando et al. 2013]) these remain quite experimental and often restricted to being a mechanism to reduce the existing cost of simulation through model simplifications and/or reduced dimensionality. We instead would be aiming to push effective resolutions and model complexity past their current limitations, striving to capture novel scales of plausible motion, which is unlikely to be properly represented with simplified adaptive methods regardless of resolution. Furthermore, even if we were able to use the use the required high resolutions, the driving forces behind interactions at smaller scales (and with multiple phases) are not always the same as those on larger scales and so models would have to be developed to account for this variation as well. These issues in increasing resolution of standard techniques have instead inspired a collection of alternative methods for splashes and sprays that decouple smaller scale behaviours from the bulk fluid motion such that they may be simulated separately and/or using different models/discretisations (secondary particle simulations, gas simulations etc).

2.4.2 Previous Approaches

Some methods have coupled free-surface models with different discretisations to model splashes with droplets and volumetric sprays. Takahashi et al. [2003] introduced the notion of coupling free-surface liquid simulations with secondary particle systems for their splashes with droplets, sprays and foam using additional components following very simple behaviours. Building on this idea, Song et al. [2005] then coupled ballistic particles for bubbles and droplets and to their Eulerian grid-based solver, preserving mass throughout the transition between states and considering different numbers of ejected droplets and bubbles dependent on the velocity of the fluid. Kim et al. [2006] similarly consider secondary particles ejected from their Marker Level-Set approach, using varying radii droplets and creating background mist/sprays dependent on the velocity such that more turbulent areas create denser volumetric effects.

Other approaches have looked to increase or improve free-surface liquid simulations to better capture splashing behaviours, particularly the breaking off of droplets, without also trying to couple volumetric components. Losasso et al. [2008] couple their PLS solver with a secondary SPH/FLIP/PIC hybrid system. In this method, they modify their pressure projection to allow varying particle density in their hybrid simulation component and use a blend of the incompressible grid velocity with the (divergent) particle velocity based on particle density to create finer droplet-like details in more sparsely sampled regions. Droplets are similarly captured by the method of Mihalef et al. [2009] who also use bubble particles to increase the visual plausibility of their Marker Level-Set free-surface liquid simulations. The break off of these is motivated by the Weber number (which we will discuss more in Chapter 4) and the marker particles used to track the surface. Gerszewski and Bargteil [2013] modify a standard FLIP/PIC simulation to allow unilateral incompressibility i.e. positive divergent velocity, which allows expansion/breaking-off of particles from the main liquid body, creating more splashy results. However the approach taken requires costly solution of two linear complementarity problems. More recently, Um et al. [2018] have looked to improve the splashing behaviour and fine-scale details of FLIP/PIC simulations using a machine-learning driven droplet break-off mechanism.

Combining elements from much of the previous work, Yang et al. [2014] develop a unified model for free-surface liquid, droplets and sprays using coupled FLIP/PIC, blended FLIP/PIC (from Losasso et al. [2008]) and volumetric spray components. This also considers interesting spray phenomena such as primary liquid break-up into droplets, drag (from Mihalef et al. [2009]) and secondary atomisation. However this method ignores interactions within the droplets and requires multiple solves for each component in the coupled system. Most importantly, the method is designed to target real-time applications and the results they demonstrate, whilst impressive, are not photorealistic.

Chentanez et al. [2015a] create a unified system for fast simulation of larger-scale liquid phenomena, coupling heightfield shallow water simulations, with grid-based and particle-based free-surface simulations for higher-quality interactive simulations but do not consider the other components such as droplets, sprays and foam.

Perhaps the most successful of all works from the computer graphics community in practice is that of Ihmsen et al. [2012] who describe a method for creating secondary ballistic particle simulations from initial particle-based free-surface liquid simulations. As they separate the simulation of initial free-surface liquids from these secondary elements, this allows the method to be used in a layering-style workflow, creating elements separately to reduce cost if changes are needed. This uses 3 metrics based on the liquid simulation such as curvature of the free-surface, acceleration and local velocity variation to determine sources for ballistic particles. These particles then gain different behaviours based on their position relative to the liquid surface such that they are able represent spray, bubbles and foam elements, and transitions between them. This is similar to many of the other methods before it but focuses on allowing use as a post-process rather than coupled into the simulation. However these components are purely driven by the input simulation and do not interact with one-another, restricting the level-of-detail in the motion captured.

In summary, a collection of approaches have looked to solve the issue of splashing liquids, often looking at coupled models with various discretisations [Takahashi et al. 2003; Yang et al. 2014; Chentanez et al. 2015a]. However, none of the model attempting to create these behaviours has targeted photorealism, instead these methods have all been developed for interactive purposes. In this way, the quality of the results that they demonstrate is not to the quality that would be required by feature-film VFX. Approaches that can be used in the layered workflow used by effects artists have been most effective and widely adopted, although not often without modification as we will see in the next sections.

As well as references from the academic community, there are examples from within industry as well that describe alternative approaches to these behaviours.

Examples from VFX

Splashing liquid simulations are a large part of most FX-heavy productions and are not solved out-of-the-box by available simulation frameworks. As such, there are a number of documented approaches from within the VFX and animation communities for their approaches to these effects. These describe some of the main ways that people have tackled these problems, and most importantly highlight the necessity that artists have found for bespoke and non-physical elements or compositing tricks in order to create production-quality results. From our experience, working alongside the artists at DNEG, it is clear that similar practices are still required to make the full range of details that are required in a liquid effect for feature film (and we will discuss this in the next section).

Documented examples from production include: combining volumetric fluid simulation with animated spray and foam clouds, compositing of live action elements and hand placed particle systems for 'Lord of the Rings' [Kurtz and Duda 2002]; secondary particle systems and pre-simulated 2D/3D splashes for 'The Day After Tomorrow'; hand-sculpted splash effects for 'Ice Age: The Meltdown' [Thornton 2006] spray, bubble and foam particle simulations and artistically-placed particle emitters for 'Ratatouille' [Froemling et al. 2007] (not photoreal but still highly detailed); huge character-driven fluid-simulations on 'The Chronicles of Narnia: Prince Caspian' [Trojansky 2008] and clustered simulations using particles and volumes for droplets, foam and froth on 'The Good Dinosaur' [Reisch et al. 2016]. Recently for 'Pirates of the Caribbean: Dead Men Tell No Tales', Hopper and Wolter [2017] also recognise that most of the time spent in making production simulations is in 'secondary' elements, noting that these are important for giving relative scale to the results.

Many recent production practices come from a combination of the works of Ihmsen et al. [2012] and Zhu and Bridson [2005] alongside the notion of blending liquid motions from Losasso et al. [2008]. Standard tools in Houdini [Side Effects 2018] for 'whitewater' implement a version of Ihmsen et al. [2012], creating simple particle systems from areas of high velocity, vorticity and curvature of the motivating liquid surface. These particles by default fall simply under gravity but due to the inherent flexibility of a tool-set such as Houdini, artists can couple

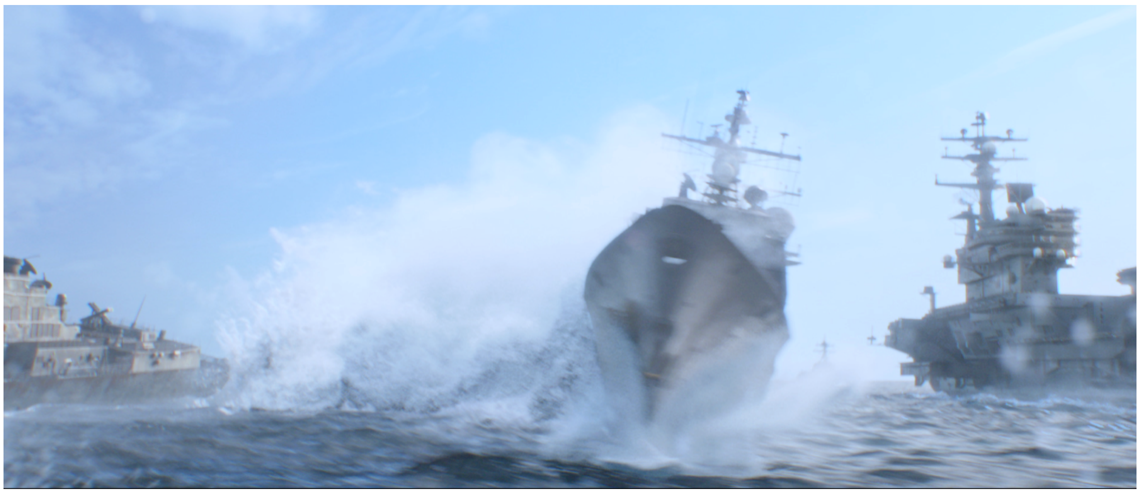


Figure 2.6: A shot from the recent film *American Assassin* [Cuesta 2017] demonstrating a liquid splash as a ship's mast crashes into the water. Effects by DNEG's Ole Eidsheim. ©2017 CBS FILMS INC. and LIONSGATE ENTERTAINMENT INC.

with air fields, noise and other influences to increase the variety captured in the result. Other less well-defined workflows we have seen use secondary FLIP/PIC simulations with a variety of external and user defined influences such as drag, noise fields and particle age. Ultimately, arising from the flexibility of the tools available to them, artists are able to combine many of the usual approaches together to create things like splashes with sprays and mists (coupled FLIP/PIC and volumetric fluid simulations) *but* to do so requires artistic time and effort as no single approach is defined that solves all the required elements.

In the next section we will look at an example workflow taken to create a plausible liquid effect, and the way in which current artistic tools are used to recreate such complex phenomena.

2.5 How to Create a Production Quality Liquid Effect

To demonstrate the efforts required by artists in the creation of production quality liquid effects we use an example from a recent film worked on by DNEG, *American Assassin*. This effect looked to model a ship's mast breaking off of a battleship and crashing into the water. The final image we can see in Figure 2.6.

The workflow taken by the artist to create the liquid splash in this example was as follows:

1. First, taking a pre-authored ocean simulation (a 2D heightfield), a FLIP free-surface liquid simulation was created to follow the movements of the ocean, but most importantly contain the falling mast and create the main splash element. The initial setup of the splash was relatively straightforward as the tools for purely free-surface liquid simulations are well-established and even things such as coupling to the movements of the ocean is handled out-of-the-box (by an approach similar to Nielsen and Bridson [2011]).

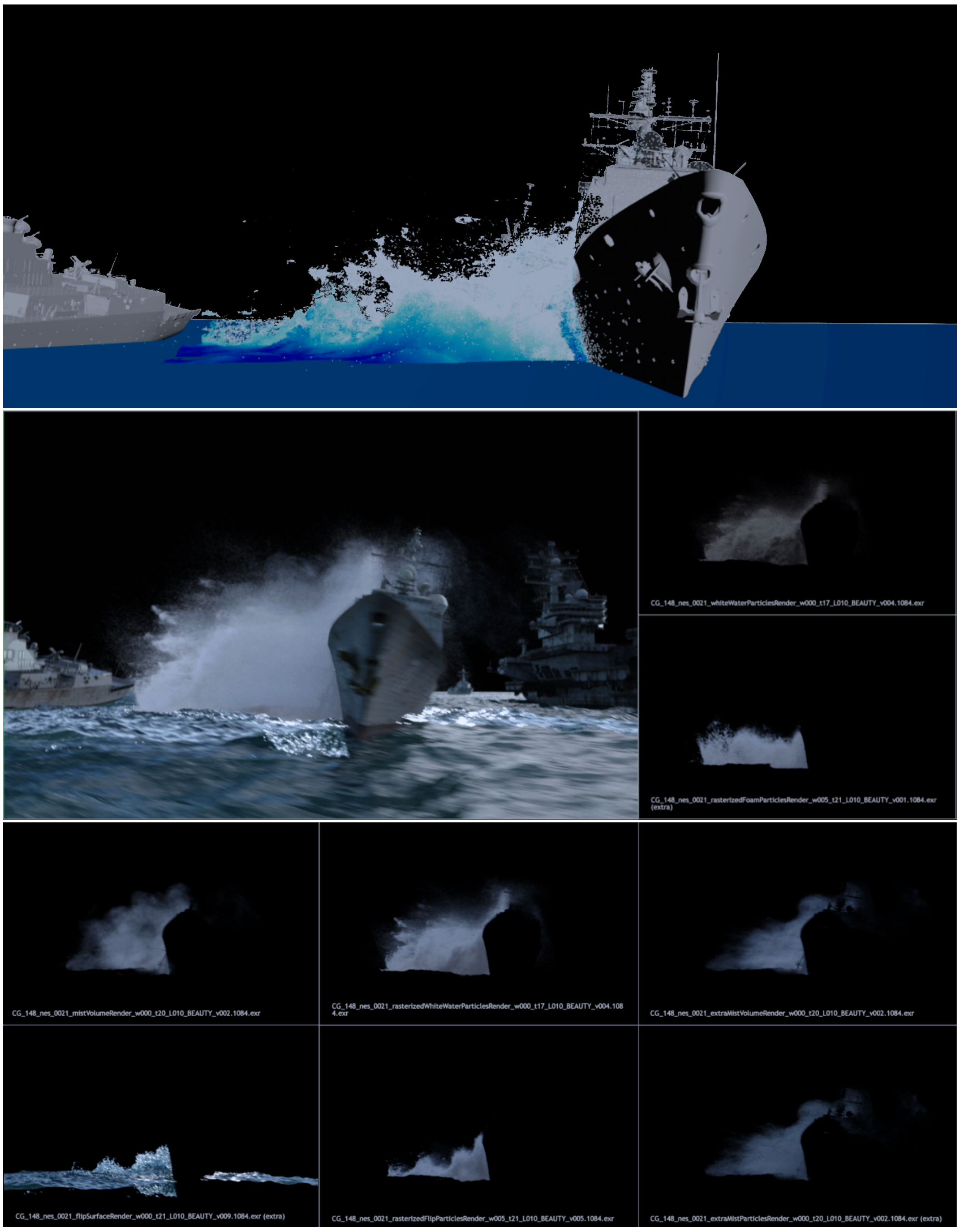

CG_148_nes_0021_whiteWaterParticlesRender_w000_t17_L010_BEAUTY_v004.1084.exr
CG_148_nes_0021_rasterizedFoamParticlesRender_w005_t21_L010_BEAUTY_v001.1084.exr (extra)
CG_148_nes_0021_mistVolumeRender_w000_t20_L010_BEAUTY_v002.1084.exr
CG_148_nes_0021_rasterizedWhiteWaterParticlesRender_w000_t17_L010_BEAUTY_v004.1084.exr
CG_148_nes_0021_extraMistVolumeRender_w000_t20_L010_BEAUTY_v002.1084.exr
CG_148_nes_0021_flipSurfaceRender_w000_t21_L010_BEAUTY_v009.1084.exr (extra)
CG_148_nes_0021_rasterizedFlipParticlesRender_w005_t21_L010_BEAUTY_v005.1084.exr
CG_148_nes_0021_extraMistParticlesRender_w000_t20_L010_BEAUTY_v002.1084.exr (extra)

Figure 2.7: Work-in-progress renders and contact sheets of some of the rendered elements created to create the splash effect in Figure 2.6. These elements include extra simulation passes for whitewater (spray, foam, bubbles), mist and vapour, as well as both point and surface renderings of the initial FLIP fluid simulation. In total 78 unique elements were passed on to the next stage of the pipeline for this effect.

2. Having assessed the initial simulation, the artist found the need to modify this simulation. After some iteration, the modifications chosen included adding a density-based wind force, quoting the artist on the shot: "for breaking up airborne fluid and add directionality to turbulent fluid" and a "custom solver for air drag based on air density, the cross-section of the particle and a numerical drag coefficient for a smooth sphere" triggered by the distance to the FLIP liquid surface and speed.

3. Once the primary splash simulation was complete, a multitude of steps for processing this simulation were taken. These involved removing points based on their lifetime, speed, density and creating masks based on various metrics such as the distance a particle has travelled since its initialisation or its speed and height. These masks were then used to modify the resulting liquid surface.

4. Given the post-processed primary splash, secondary simulation passes were also required for finer-scale and multi-phase elements "as the initial splash was very energetic and created unwanted features like strings and blobs especially close to the ship on mast impact, most of the work involved with the whitewater was manual shaping and clean-up after sim". Similar to the ocean-splash setup, methods such as Ihmsen et al. [2012] are implemented in Houdini and configured to easily work with the initial liquid simulation output (even when it has been post-processed).

5. However, similar to the required intervention in the primary splash simulation, the artist was required to further make a collection of manual changes and additions to this secondary particle setup to achieve their desired result. These included: removing low density particles from the secondary simulation sources to avoid droplets from the primary simulation creating further droplets in the secondary simulation; removing fast particles and manually removing particles using bounding geometries and volumes; detecting clumps and applying noise based on height in the domain; and finally, applying drag and enforcing a soft speed limit on secondary particle movement.

6. Following this, multiple volumetric simulations were then created using grid-based fluid simulations sourced from these secondary particles for various mist elements.

7. Finally, all of the elements were rendered in various forms, with FLIP liquid simulations rendered both as a surface (after application of the masks described earlier) and as particles, and secondary particle simulations rendered in various passes breaking up different states (spray, foam, bubbles) as both points and volumes as well.

In total, with all of the various components, and the different versions of these components required to get "enough density and detail", it meant that the artist delivered 78 different rendered elements downstream to create the final shot. Whilst it is likely that not all of these were used in the final image seen in the film, that does not reduce the amount of work required by the FX artist to create them, especially considering the unspecified number of iterations that the artist would have had to create to find correct parameter values/additional forces or modifications to add.

This example demonstrates an extreme, albeit not unprecedented, amount of work required by an FX artist to a create production quality splashing liquid effect. Commonalities with this example and the many other documented workflows from the previous section can be seen in things like the use of different layers of secondary elements, of differing or similar behaviours; the requirement for artists to modify and tweak even the physically-based components of their effects such as the primary liquid simulation; and the importance of artistic iteration and control in defining the required modifications to reach their desired result.

2.6 Areas for Improvement

Whilst many good examples of plausible liquid phenomena have been created using these existing workflows, it is clear that approaches that are so reliant on artistic input for the creation of high levels-of-detail and plausibility are extremely costly in terms of both artist time and computational resources i.e. CPU time for iterative simulations, storage and rendering of separate elements and their different versions etc. Following our investigation into these workflows above, we believe there are a few main areas of improvement that stand out as ways to mitigate some of the issues with this approach. These are:

- Efficiency and performance - as resolutions and demands on detail increase, so do computational costs. Coupled with the iterative workflow of these kinds of effects, this can become even more expensive. Are there ways to reduce the computational cost of simulations to facilitate these increasing demands?

- Level of detail - whilst increasing resolution of simulation is a generally accepted way to improve detail, we have seen that it is not only resolution that provides the required details in practice. In production quality liquid effects, many important details are provided by secondary simulations. Are there ways of improving these to include higher levels of detail such that the required number of these simulations could be reduced? Similarly, we have seen that the use of secondary simulations often makes up for many of the issues with the primary simulations not demonstrating required behaviours such as droplet break-off. This is particularly true for splashes in liquid simulations where fine details can be lost through post-processes such as creation of a liquid surface. Are there improvements to be made to allow the best possible use of these initial physically-based simulation results?

- Range of phenomena - the current tools have been shown to be very good at simulating certain things such as larger scale liquid motions and liquids with well-defined liquid/air interfaces, yet their handling of splashing liquid and ‘whitewater’ phenomena is less well-established, placing requirements on artists to create these elements. Can the range of phenomena in physically-based simulations be expanded to include more of these scenarios?

- Artistic control - the most important aspect of all simulation frameworks for visual effects production is how an artist can interact with it. Are there ways in which we can improve the usability and control over simulations, and their resulting data, to better facilitate the inevitable artistic intervention in these workflows?

The next chapters of this thesis aim to tackle problems in these areas, looking to increase the quality and turnover of difficult liquid effects in VFX production. Building upon existing techniques, we look to introduce new developments into existing workflows, improving artistic interaction with their tools, and helping with the simulation and post-processing of liquid effects.

Chapter 3

Increasing Iteration: Fluid Re-Simulation using Model Reduction

This project documents an initial investigation into increasing iteration on fluid simulations, shown to be an issue in Chapter 2 and represents a useful review and grounding for future work in this area. However on changing industrial partners, a greater issue and possible area for contribution was identified in the detail and control of high-end effects, taking priority in this research project. That being said, due to the academic and potential future production contribution here it is included in this thesis.

3.1 Overview

As discussed in the last chapter, the art of authoring a simulation is an extremely iterative process. Scenes and setups are often laid out by senior artists for other artists to iterate upon. Initial simulations can then be run with a multitude of parameters and input sets before an artist decides on the general setup and runs subsequent iterations with minor tweaks and additions upon this, until their results exhibit desired behaviour and are approved. Usually, each iteration will require a new simulation to be run and cached to disk but, as these changes are often quite minor, there is a lot of time spent running very similar calculations to those of a previous iteration. In this way, each simulation creates a massive amount of data, only for this to be ignored when running the next iteration. Our work here focuses on re-simulation, looking to utilise the work done and data created in performing previous iterations of a fluid simulation to reduce that required for subsequent iterations[1].

The problem of speeding up the simulation process in the general case remains an important challenge for researchers. We take the following approach in the hope to be largely orthogonal to these methods and to allow it to be used in tandem with these other developments, aiming to enable even faster fluid simulations for the special case of re-simulation. In this chapter,

[1]This part of the research project was originally undertaken with the hope to allow an extension of any data-driven re-simulation methodology to the case of liquid simulations. However the methods used have not displayed the flexibility required for this purpose. As such, we document this work in the context of general grid-based fluid simulations and do not consider the added difficulties required in liquid free-surface simulations.

we describe our investigation into using data-driven model reduction as a fast re-simulation system for fluid simulations.

The most expensive part of a fluid simulation is often the pressure projection (Equation 2.5), as such we focus the model reduction approach to this stage in the simulation. This will also allow integration into any simulation frameworks that use a grid-based pressure projection and avoid unnecessary impact (i.e. reduced fidelity/range of behaviour) on less expensive parts of the simulation.

We will first show how to use data-driven model reduction to speed up calculations in an Eulerian pressure projection and then go on to explain the problems that we face when applying this to fluid simulations for VFX production, with discussion of some possible extensions and solutions detailed at the end of the chapter.

3.2 Model Reduction

Model reduction is used to reduce the dimensionality of a problem with many degrees of freedom so that it can be more easily solved, whilst preserving behaviour from the original high dimensional model. Reduced systems generally require far less computation than their higher dimensional counterparts, greatly increasing the speed of solutions to problems in the reduced space. Here we describe a projection-based model reduction approach to our fluid simulation.

3.2.1 Projection

The general concept of projection is that of an injective mapping P between the space X and a subspace $Y \subset X$ such that

$$P : X \to Y \tag{3.1}$$

and $P \circ P = P^2 = P$.

In Chapter 2 we demonstrate the use of a projection operator to enforce incompressibility, mapping our velocity vector field $\mathbf{u} \in \mathbb{R}^N$ onto the subspace of divergence-free vector fields, say $\mathbb{D}^N \subset \mathbb{R}^N$.

Here, we instead consider the case of projection $\mathbb{P}$ of some state vector $\mathbf{s} \in \mathbb{R}^N$ onto a subspace that has lower dimensionality, say $F \subset \mathbb{R}^r$ (for $r < R$), such that

$$\mathbb{P} \colon \mathbb{R}^N \to F \tag{3.2}$$

$$\mathbf{s} \to \mathbb{P}(\mathbf{s}) = \tilde{\mathbf{s}} \in \mathbb{R}^r \tag{3.3}$$

For our purpose we also require the back-projection operator $\mathbb{Q}$ such that

$$\mathbb{Q} \colon F \to \mathbb{R}^N \tag{3.4}$$

$$\tilde{\mathbf{s}} \to \mathbb{Q}(\tilde{\mathbf{s}}) = \hat{\mathbf{s}} \in \mathbb{R}^N \tag{3.5}$$

As we will see in §3.2.2, if we have ways of performing our simulation steps in the reduced state, which has lower dimensionality than our full state, we can greatly speed up simulation time. In this way it is possible to either choose to reduce our entire fluid system to work in the reduced space [De Witt et al. 2012; Treuille et al. 2006] or allow specific operations to be performed in the reduced space and back-project the result into the full space for other operations [Ando et al. 2015; Kim and Delaney 2013].

Current work in model reduction for fluid simulation has focused on using simple linear projection operators (see below) for the reduction of dimensions. Due to the huge scale of systems that are being handled in fluid simulations it is considered unfeasible to use more advanced, nonlinear projection formulations. This is because the precomputation stage used to extract linear bases for large-scale systems is already a very costly step, even before considering more expensive nonlinear methods, especially for data-driven models. Similarly, to effectively use this technique to increase efficiency, the projection and back-projection must be relatively low cost operations, which is not the case for nonlinear methods. Therefore we continue to focus on the case of linear projection in this work.

Linear Projection

For the case of a linear projection operator, this is equivalent to approximation of $\mathbf{s} \in \mathbb{R}^N$ as a linear combination of basis vectors $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$,...$\mathbf{x}_r \in \mathbb{R}^N$ such that:

$$\mathbf{s} = \mathbf{X}\tilde{\mathbf{s}} \tag{3.6}$$

where $\mathbf{X} \in \mathbb{R}^{N \times r}$ is a linear projection matrix containing the r basis vectors $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$,...$\mathbf{x}_r \in \mathbb{R}^N$ as columns and $\tilde{\mathbf{s}} \in \mathbb{R}^r$ is the reduced space analogue of $\mathbf{s} \in \mathbb{R}^N$ containing coefficients for each basis vector.

Now if these basis vectors are orthonormal (i.e. $\mathbf{X}^T\mathbf{X} = I_d$, the $n \times n$ identity) then we can easily calculate

$$\tilde{\mathbf{s}} = \mathbf{X}^T\mathbf{s} \tag{3.7}$$

the projection of a full dimensional vector $\mathbf{s} \in \mathbb{R}^N$ into its reduced state $\tilde{\mathbf{s}} \in \mathbb{R}^r$ where $r << N$. Similarly we can back-project this reduced vector to a full dimensional vector using Equation 3.6.[2]

We will now describe how this system has been previously used in application to fluid simulation.

3.2.2 Related Work

The original application of model reduction to fluid simulation was for the study of turbulent flows by Lumley [2007] and since then has been used for fluid dynamics in various engineering and mathematics applications [Berkooz et al. 1993; Krysl et al. 2001].

Model reduction was introduced to the computer graphics community by Pentland and Williams [1989]. Following initial use in solid simulation [Barbič and James 2005], Treuille et al. [2006] used model reduction to interactively simulate Eulerian fluids. By using a lower dimensional simulation on a space defined by divergence-free vector fields they remove the pressure projection stage from their simulations entirely. The basis for this space is extracted

[2]The vector $\hat{\mathbf{s}} = \mathbf{X}\mathbf{X}^T\mathbf{s}$ may not be the original state vector as the projection to the reduced state is only optimal in a least squares sense.

from a set of high resolution fluid 'snapshots' using the method we describe in detail in §3.2.3. Their model allows simulation of relatively simple fluid effects in real-time with interactivity through moving collision objects. Wicke et al. [2009] build on this work, adding the ability to tile simulation domains in order to create larger-scale reduced simulations. However, the system used for changing boundary conditions and collision objects in this work requires lengthy precomputations and the simple finite difference advection scheme used in the reduced space causes dissipation of energy throughout their simulations. Replacing the dissipative advection operators from Treuille et al. [2006], Kim and Delaney [2013] use cubature to approximate Semi-Lagrangian advection in the reduced space and create a reduced model able to closely recreate more complex full dimension simulation results for smoke simulations. Their work introduces the use of model reduction as a technique for re-simulation and begins to describe the effect of parameter change in subsequent iterations which we explore further in this work.

As we will discuss further later, the reduced basis used is very important to the quality of results and flexibility of the reduced model. This was the main subject of the work by Gerszewski et al. [2013] who develop a method to add arbitrary new basis elements, such as those designed by an artist or created by curl-noise [Bridson et al. 2007].

Rather than extracting a reduced space from prior fluid states, De Witt et al. [2012] calculate a reduced basis through eigenvectors of a Discrete Exterior Calculus formulation of the Laplacian for their fixed fluid domain. Through the constraints imposed on their Laplacian they ensure that the velocity fields are divergence-free and satisfy the boundary conditions required. A similar approach has recently been taken by Liu et al. [2015]. However, as our motivation for the use of model reduction is to utilise existing data from previous simulations, we focus on improving the data-driven methods above.

Ando et al. [2015] describe an alternative method of reducing the dimensionality of a pressure projection for liquid simulations. They use a 'surface-aware' trilinear interpolation kernel for reducing their system that allows application to an evolving liquid free-surface, but, causes smoothing of pressure values leading to volume loss and energy dissipation. We use a similar approach, focusing on the pressure projection, but instead use data-driven model reduction, aiming to retain the fidelity of the higher order method and small scale details as in Kim and Delaney [2013], at the cost of some precomputation. This decision was made so that we may offer a means to increasing turnover of simulations without causing obvious differences between our full resolution simulation and our reduced simulations. We do not wish to use a method that always causes a significant difference or decrease in quality, such as the seams and energy loss/smoothing present in Ando et al. [2015]. The data-driven methods are designed to retain the level of detail of the input simulations, to allow greater fidelity than using an interpolation method.

3.2.3 Proper Orthogonal Decomposition via Method of Snapshots

The key component of all of the above reduced order methods [Ando et al. 2015; De Witt et al. 2012; Kim and Delaney 2013; Treuille et al. 2006; Wicke et al. 2009] is the projection operator defined by the matrix **X**, that projects the system onto a low order subspace that aims to encapsulate the behaviour of our original high order model.

As shown in Treuille et al. [2006] (and used in Wicke et al. [2009] and Kim and Delaney [2013]), it is possible to extract a suitable basis, defining $\mathbf{X}$, from a collection of previously calculated fluid states using Proper Orthogonal Decomposition (POD) via the Method of Snapshots. This method is effectively equivalent to performing Principal Component Analysis (PCA) on the entire simulation, but here we use terminology consistent with the engineering literature and Kim & Delaney's work on re-simulation [Kim and Delaney 2013]. We will now describe how to use this method to extract the projection matrix $\mathbf{X}$.

First concatenate the state vectors, say the pressure states[3] out of the pressure projection (Equation 2.18) $\mathbf{p}_i$, into a state matrix of snapshots $\mathbf{P} = [\mathbf{p}_1|...|\mathbf{p}_t] \in \mathbb{R}^{N \times t}$ where $\mathbf{p}_i \in \mathbb{R}^N$ is the full dimensional state vector describing our system at timestep i and t is the final timestep of the input simulation.

Now perform Singular Value Decomposition (SVD) on this matrix to get:

$$\mathbf{P} = \mathbf{X}\Sigma\mathbf{Y}^T \tag{3.8}$$

where $\mathbf{X} \in \mathbb{R}^{N \times t}, \Sigma \in \mathbb{R}^{t \times t}$ and $\mathbf{Y} \in \mathbb{R}^{t \times t}$, $\mathbf{X}$ and $\mathbf{Y}$ are matrices with orthonormal columns and Σ is the diagonal matrix of *singular values* that represent the relative contribution of each column in $\mathbf{X}$ to the simulation state matrix.

From this we can take the columns of $\mathbf{X}$ as our orthonormal basis that spans a lower dimensional subspace. These basis vectors $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_t$, known as Proper Orthogonal Modes (POMs), describe the key common components of the simulation data contained in the state matrix. For pressure field snapshots we find the most common components used to correct the divergent velocity fields to be divergence-free (Figure 3.2). In the case of velocity field snapshots, the POMs describe the key common components of velocity field over the entire input simulation (Figure 3.3). Furthermore, if we use an input of the post pressure-projection velocity fields, our velocity POMs will be divergence-free and enforce the boundary conditions of our simulation [Treuille et al. 2006].

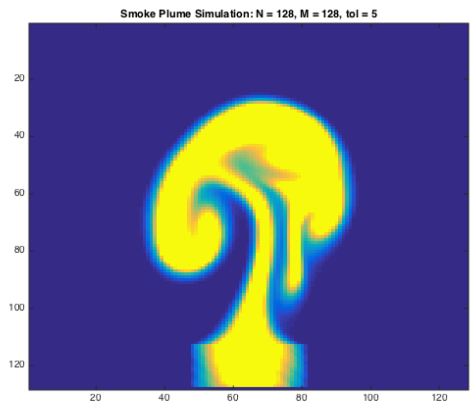


Figure 3.1: Single frame of 128 × 128 smoke plume style simulation

[3]This method can be used for others state in our simulation, e.g. the velocity $\mathbf{u}$.

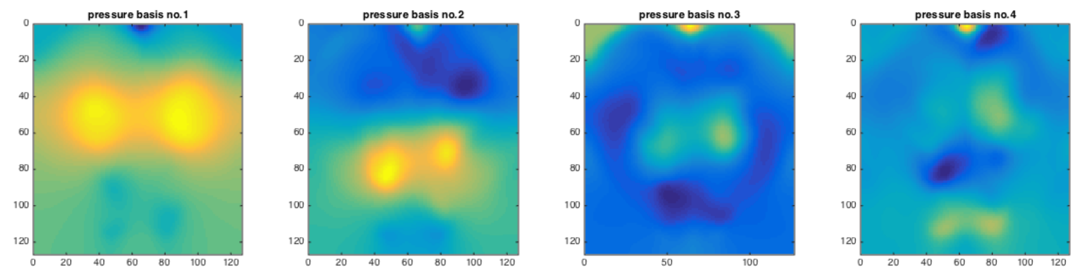


Figure 3.2: The first 4 elements $\mathbf{x}_1, ..., \mathbf{x}_4$ of our pressure basis $\mathbf{X}$ for the 128^2 sim in Figure 3.1.

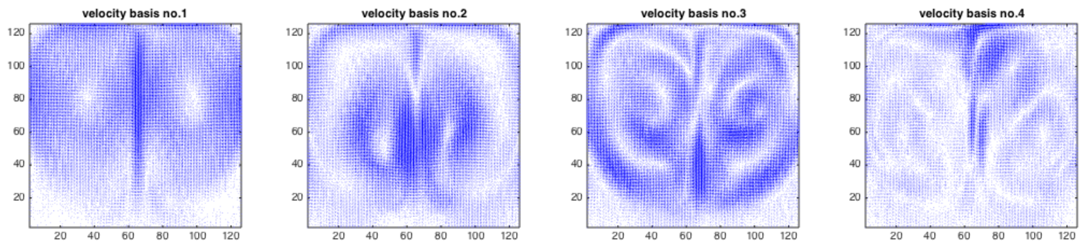


Figure 3.3: The first 4 elements $\mathbf{v}_1, ..., \mathbf{v}_4$ of our velocity basis $\mathbf{V}$ for the 128^2 sim in Figure 3.1.

This SVD is such that the basis defined by the columns of $\mathbf{X}$ is the best (in the least squares sense) fit for representation of the original snapshots in a reduced space. That is, any subspace spanned by any k POMs $\mathbf{x}_1, ..., \mathbf{x}_k$ for $1 \leq k \leq r$ is the best fit k-dimensional subspace of our state matrix $\mathbf{P}$. Using this result, the number of POMs can be reduced to increase speed and reduced memory footprint at the cost of range of motion available in the subsequent reduced simulation, whilst remaining the best possible fit to the input simulation for that number of basis elements. The most popular method for deciding which POMs to remove uses the corresponding singular value from Σ (using Equation 3.8), if this is lower than some threshold e.g. $1e^{-9}$, we can say that the POM has very little influence in the simulation and can be discarded.

3.3 Subspace Pressure Projection using Previous Simulation Data

The use of data-driven model reduction for re-simulation by Kim and Delaney [2013] and restriction of dimensionality reduction to the pressure projection by Ando et al. [2015] were the main inspirations for our work. As we mentioned in Chapter 2, there is a bottleneck in the FLIP/PIC simulation framework at the grid-based pressure projection that greatly impacts on overall simulation time. Here, we combine these ideas to show that we can increase the efficiency of our simulation using data-driven model reduction confined to the pressure projection.

Kim and Delaney [2013] originally used repeated projection and back-projection to and from their reduced state to allow application of forces at full resolution, Ando et al. [2015] then developed a system using this idea to restrict use of model reduction to only the grid-based pressure projection. However, Ando et al. [2015] suffer from volume loss and smoothing of resulting fluid velocities as they use trilinear interpolation for their model reduction. Our work aims to retain the fidelity of the re-simulated results originally in Kim and Delaney [2013] in the same self-contained framework as Ando et al. [2015].

By projecting the simulation into a reduced state before the pressure projection and then back-projecting into a full state afterwards we can perform the rest of our simulation timestep at full resolution. Doing this, our reduced model is fully self-contained to the pressure projection step and allows use of full accuracy at the other, less computationally expensive stages of our simulation. Therefore, our reduced pressure solve should be applicable to fully Eulerian solves and also, importantly for VFX, hybrid simulations such as FLIP/PIC. Another benefit of using only a reduced pressure projection stage in our simulations is that we reduce the huge amount of precomputation required by Kim and Delaney [2013] (4 SVDs and a cubature calculation) to a single (or two, see Equations 3.13 and 3.14) SVD(s).

3.3.1 Method

The pressure projection step is solved through the solution to a large linear system (Equation 2.18) of size $N \times N$:

$$\mathbf{A}\mathbf{p} = \mathbf{b}$$

If we suppose that $\mathbf{p} = \mathbf{X}\tilde{\mathbf{p}}$ and $\mathbf{X}^T = \mathbf{X}^{-1}$ we can reduce this system into that of solving a much smaller $r \times r$ linear problem through Galerkin projection [Stanton et al. 2013]:

$$(\mathbf{X}^T\mathbf{A}\mathbf{X})\mathbf{X}^T\mathbf{p} = \mathbf{X}^T\mathbf{b} \tag{3.9}$$

So our reduced pressure solution can be calculated by

$$\tilde{\mathbf{p}} = \mathbf{X}^T\mathbf{p} = (\mathbf{X}^T\mathbf{A}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{b} \tag{3.10}$$

This was the technique used by Kim and Delaney [2013] and Ando et al. [2015] in their reduced fluid simulation systems. We now back-project this into a full resolution solution as in Ando et al. [2015]:

$$\hat{\mathbf{p}} = \mathbf{X}(\mathbf{X}^T\mathbf{A}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{b} \approx \mathbf{p} \tag{3.11}$$

and the pressure gradient update to remove divergence is given by

$$\hat{\mathbf{u}} = \mathbf{u}_{preproject} - \frac{\Delta t}{\rho}\nabla\hat{\mathbf{p}} \tag{3.12}$$

Unfortunately, when solving in this way, we found that approximation errors accumulated causing divergence to grow as the simulation progresses (Figure 3.4), even when using a few subsequent Jacobi iterations on our approximate full resolution solution as in Ando et al. [2015]. We believe this to be the cause of the volume loss that they experience, which they later correct with the volume loss correction scheme from Kim et al. [2007]. This is possibly be due to the fact that by having an overdetermined system (as usually $N >> t$) and discarding

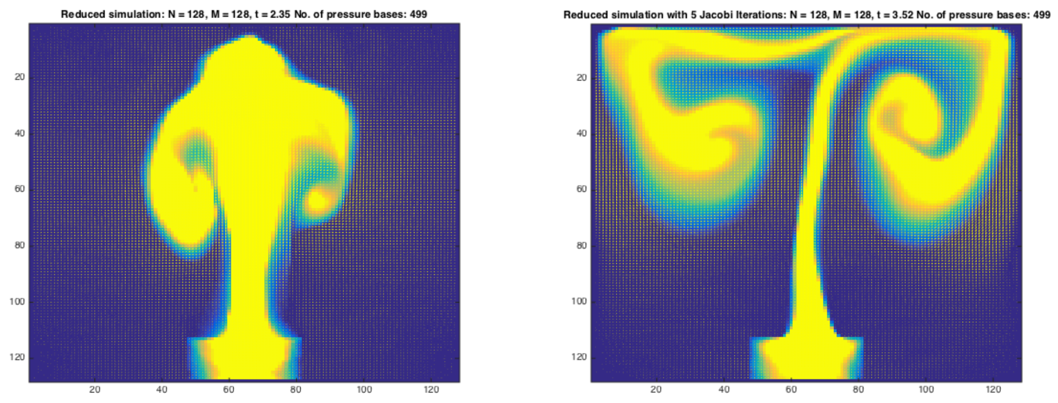


Figure 3.4: Simulation of 128^2 smoke plume using reduced pressure solution (Equation 3.12) only. Demonstrating visible accumulation of divergence (left), with addition of 5 Jacobi iterations (right).

bases with low singular values, our projection matrix is only optimal in a least squares sense, meaning our projections are not exact, allowing some small divergences to persist.

We have found that using a subsequent projection of the velocities onto a divergence-free velocity basis **V** greatly decreases this accumulation of divergence. This **V** is extracted in the same way as the pressure basis (with POD), instead using the post-projection divergence-free velocity state vectors (Equation 2.19) from our original simulation. We call this method *Reduced Pressure Solve with Projection*:

$$\hat{\mathbf{u}} = \mathbf{V}\mathbf{V}^T(\mathbf{u}_{preproject} - \frac{\Delta t}{\rho}\nabla\hat{\mathbf{p}}) \tag{3.13}$$

where we have the projection matrix $\mathbf{V}\mathbf{V}^T \neq \mathbf{I}$. This inequality arises as we again have an over-determined system on which we apply our SVD and so the matrix **V** is only optimal in the least squares sense and is not row orthogonal. This projection is such that, for each divergence-free velocity mode of the basis $\mathbf{v}_i$, we find the component of the current velocity field **u** in the direction of this mode and add that to our subspace representation $\tilde{\mathbf{u}}$, i.e. $\tilde{\mathbf{u}} = \sum_{i=1}^{r} < \mathbf{u}, \mathbf{v}_i > \mathbf{v}_i$ or simply $\tilde{\mathbf{u}} = \mathbf{V}^T\mathbf{u}$. Then project back into the full space by $\hat{\mathbf{u}} = \mathbf{V}\tilde{\mathbf{u}}$, giving us a resulting divergence-free velocity field close to the original velocity field, where the divergent components of **u** have been removed as they are not spanned by **V**.

Referring back to Equation 2.19, we see that the subtraction of the pressure gradient from our velocity field comes from the application of our orthogonal projection operator $\mathbb{P}$ which maps our velocity onto its divergence-free component, or its closest point on the subspace of all divergence-free velocity fields. In Equation 3.13, we instead use our subspace projection operator $\mathbb{Q}$ s.t. $\hat{\mathbf{u}} = \mathbb{Q}\tilde{\mathbf{u}} = \mathbf{V}\mathbf{V}^T\tilde{\mathbf{u}}$ mapping our velocity field onto its closest point on the subspace of all divergence-free velocity fields that is spanned by our POMs. Following this, we propose if our velocity POMs span the entire subspace of divergence-free velocities that arise in our simulation, then this projection $\mathbb{Q}$ should be equivalent to the original orthogonal projection $\mathbb{P}$, as they both send the input velocity field to its closest point on the subspace of divergence-free velocities. Thus removing the need to do a linear solve at all. This idea is similar to Treuille et al. [2006]'s removal of divergence through restriction to simulating on divergence-free velocities, but here we remove the need to formulate all of the other operations in the subspace as well. We call this method *Direct Projection*:

$$\hat{\mathbf{u}} = \mathbf{V}\mathbf{V}^T \mathbf{u}_{preproject} \tag{3.14}$$

In the following we demonstrate that this does in fact lead to results that are very close to those that also use a reduced pressure solve, therefore offering a new, far simpler method for performing an approximate data-driven pressure projection.

3.3.2 Results

In this section we will look at the results of tests run with these subspace pressure projection operators, the *Reduced Pressure Solve with Projection* (Equation 3.13) and the *Direct Projection* (Equation 3.14). For simplicity these have been run as part of a 2D fully Eulerian smoke simulator with Semi-Lagrangian advection as in Stam [1999] using a Incomplete Cholesky PCG linear solver, in MATLAB. This simple test is chosen to restrict the focus of the tests to our changes in the pressure projection stage.

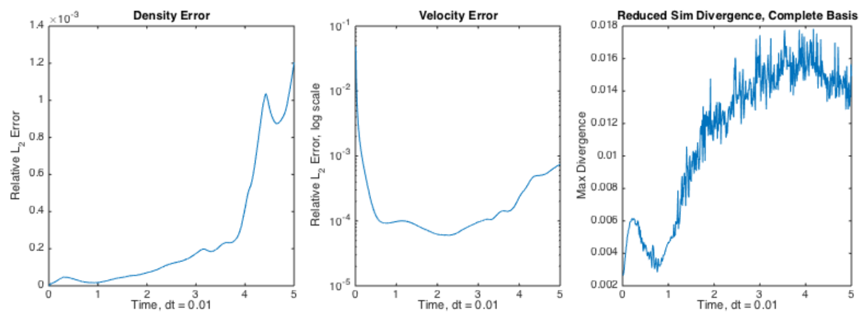
The main example simulation used for Figures 3.5 - 3.8 is the smoke plume shown in Figure 3.1, this uses a 128×128 grid with Conjugate Gradient tolerance $1e-5$. Similar to Kim and Delaney [2013], we consider the relative L_2-error as the metric for the difference between a full resolution simulation and our reduced methods.

In Figures 3.5a and 3.5b we consider the error in the density field and velocity field, and also show the total divergence of reduced simulations compared to full resolution simulation. These figures show results with a complete POM basis extracted by the SVD and Figures 3.5c and 3.5d show those using a reduced number of POMs. Table 3.1 shows the difference in times between our full solve, reduced pressure solve with projection and direct projection.

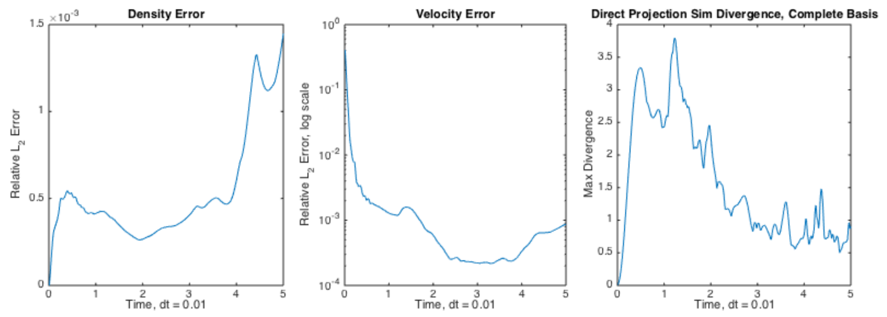
Method	*Total Time*	*Step Time*	*Speed up*
Full Pressure Solve via ICPCG	64.221s	0.128442s	
Reduced Pressure Solve & Project, complete basis.	9.157s	0.018314s	7x
Reduced Pressure Solve & Project, truncated basis.	6.1348s	0.012269s	10x
Direct Projection, truncated basis.	2.789s	0.005578s	23x
Velocity SVD	26.9837s		
Pressure SVD	8.8368s		

Table 3.1: Table of Pressure Solve times with different pressure projection operators for 128^2 smoke plume simulation in Figure 3.1.

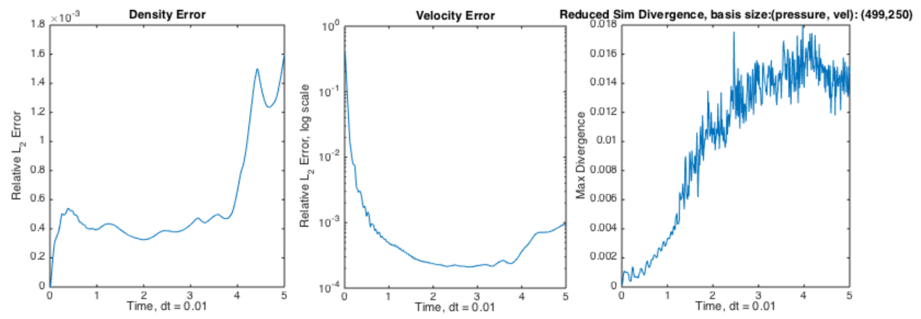
As our reduced simulations are designed to be performed as part of a re-simulation framework, we require these methods to allow the use of parameter change so that updated simulations can be created. We show the error caused by the reduced simulation when varying a simulation parameter away from the input parameter in Figure 3.7, here we have used buoyancy as our example, this could be some other global force like gravity or another parameter such as vorticity. An example of the differences created using a reduced simulation based off of a different input parameter are shown by the final states of a full and reduced simulation in Figure 3.8.



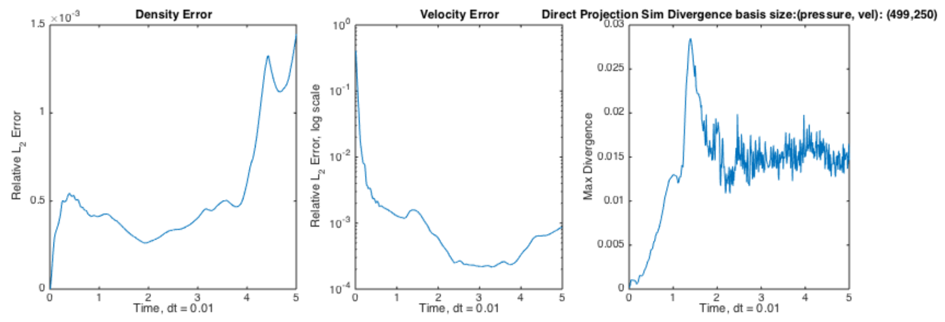
(a) Reduced Pressure Solve with Projection, complete POD basis.



(b) Direct Projection, complete POD basis.



(c) Reduced Pressure Solve with Projection, truncated POD basis.



(d) Direct Projection, truncated POD basis.

Figure 3.5: Comparisons of errors in simulations using subspace pressure solves against those using full pressure solves for the simulation in Figure 3.1. Using relative L_2-error.

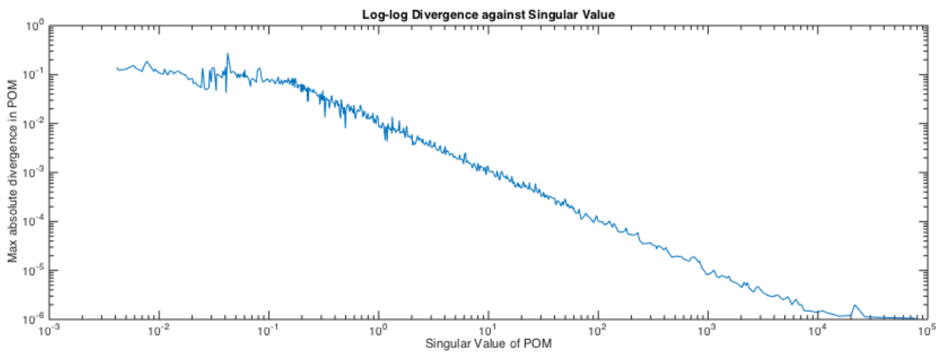


Figure 3.6: Log-log plot of divergence in the velocity POM vs singular value of that POM for 128^2 smoke plume simulation.

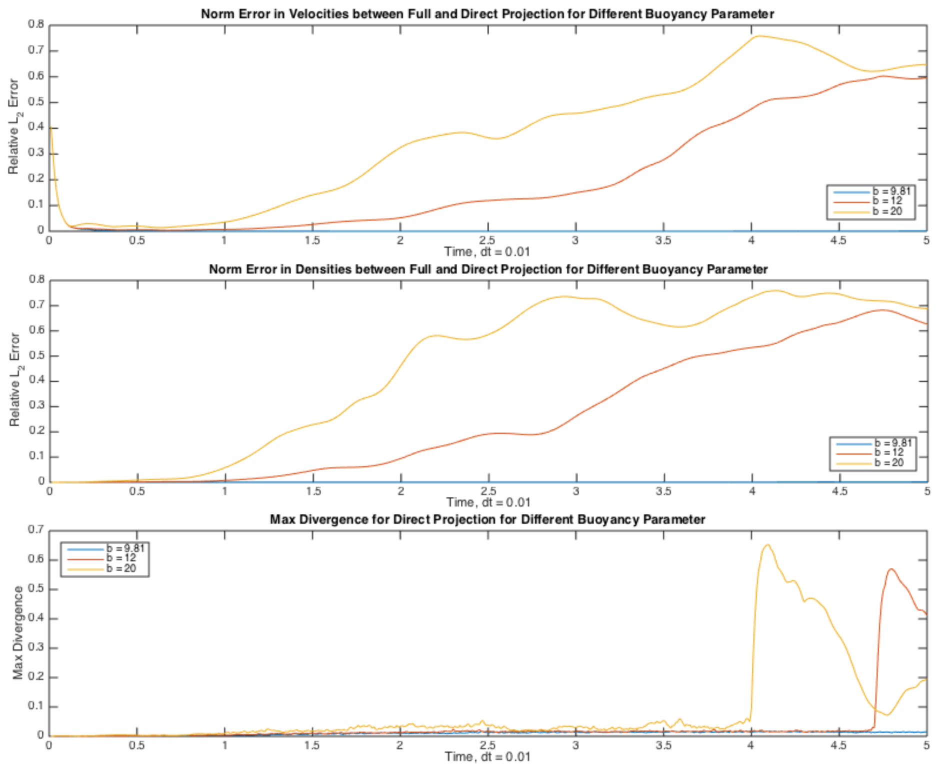


Figure 3.7: Error and max divergence when varying the buoyancy parameter in the 128^2 Reduced Simulation using only Direct Projection. Buoyancy for the input simulation $b = 9.81$.

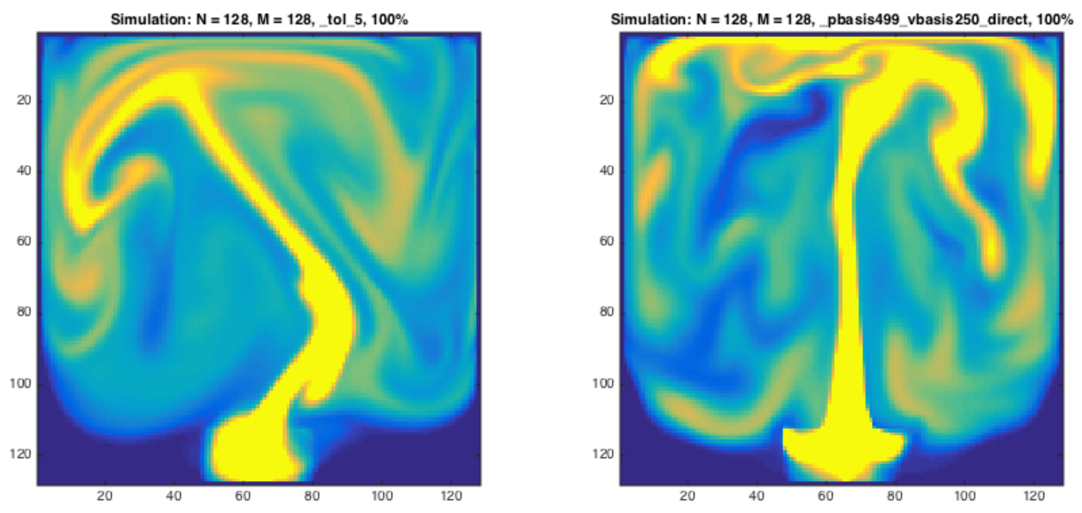


Figure 3.8: Final state of a simulation with buoyancy parameter $b = 20$. Full solve (left) and *Direct Projection* solve with $b = 20$ (right) where input basis simulation has value for $b = 9.81$. Note the large variation in the final state between these two simulations.

3.3.3 Analysis

From the figures above, we can see that it is possible to reconstruct the input behaviour to a close approximation when using the model reduced pressure projections and in doing so can offer a substantial speed increase for re-simulation (Table 3.1) due to the reduced complexity of the calculations in subsequent simulations. The results of Figure 3.5b show that using a full basis with the Direct Projection method can generate high divergences and Figure 3.6 suggests that this is due to accumulation of divergent noise in the later POMs. However it should be noted there is still only relatively small error in the velocities and densities in this case. By instead reducing the number of POMs to those with higher singular values (and so, by Figure 3.6, lower divergences), Figures 3.5c and 3.5d show that we can reduce the error in the Direct Projection method such that it is comparable to the more complex Reduced Pressure Solve with Projection. The average times for the solve in Table 3.1 suggest that the speed increase and lower pre-computation time of the Direct Projection method make it much more lucrative, making up for the slightly lower accuracy of this method. The results suggest that, as both of these methods give similar results, if we require the additional velocity projection onto the subspace anyway, the added work of projecting and separating the pressure solve in the reduced space would not be worthwhile.

Concerning the above-tolerance divergences in Figure 3.5a, even for unchanged input parameters, the results in Figure 3.6 suggest that they arise from the formulation of the updated velocity as linear combinations of the basis vectors which may have some non-zero divergences. In our examples, they generally have the same tolerance for divergence as the full scale velocities from the input simulation. As each basis element has divergence of at least the order 1×10^{-5}, large numbers of these (our coefficients of each POM in $\hat{\mathbf{u}}$) in combination could lead to the divergences that are seen in Figure 3.5b. However the results of the density and velocity field suggest this may not have a distinct impact on the rest of the simulation as each timestep is independent in its direct velocity projection and so the error in the resultant velocities and densities does not accumulate.

Unfortunately, as we can see from Figure 3.7, parameter variation away from the input simulation can lead to large differences between the reduced and full simulations. After 500 frames the density and velocity fields from the reduced simulation are both 70% different to the motion of a full resolution simulation with the same parameters. However it should be noted, similar to Kim and Delaney [2013], the resulting simulations generally remain stable and still offer predominantly fluid-like results, shown in Figure 3.8, although with different behaviours. This is the case for the majority of the simulation but this method is not always entirely robust, as seen in Figure 3.7 at the later stages of the simulations, we see very large divergence values. This is found to arise when the smoke plume drops and intersects the density source, motion not found in the original input simulation.

In summary, these results show that generally this method is only closely accurate for re-creating an input simulation from its the set of POMs. This could be useful for some applications that we had not previously considered such as storing POM caches of simulations rather than full results (this would give a reduced time required to recreate the results of a full resolution simulation but remove the need to store full simulation caches long term). However, ultimately this method, in its current form, does not give the flexibility required to use these POMs for re-simulation in VFX with varying inputs. This will be the focus of the next section, describing a few tested and as-yet untested approaches to increase the flexibility of this model reduction method.

3.4 Improvements to Model Reduction for Re-Simulation

As we have seen from the previous section, although model reduction is able to create a good increase in speed for simulations similar to that of the input simulation, there is a fundamental lack of flexibility towards parameter change in these data-driven approaches. This is unfortunately an inherent problem to many model reduction systems that use the POD Method of Snapshots. As such, there has been a lot of work in developing models to increase the flexibility of the model and the range of motion available in the POD basis [Farhat and Amsallem 2008; Vetrano et al. 2012]. We will now explore some of these previously documented approaches for parameter variation and look at other improvements we can make to this model for extension towards to a useful re-simulation system.

3.4.1 Full Dimension Pressure Projection with a Reduced Preconditioner

Preconditioning using Subspace Pressure Solve

One method that we have explored, originally developed by Jiang [2014], was to consider the above subspace pressure solve as a preconditioner for a normal PCG solve pressure projection, hoping to reduce the number of iterations required. Thus using the POMs to guide the solver in the right ‘direction’ but let the solution differ if our POMs do not solve the problem closely. Unfortunately, as shown in Jiang [2014], we cannot simply use a preconditioner comprised of only our subspace pressure solve operator matrix:

$$\mathbf{X}(\mathbf{X}^T\mathbf{A}\mathbf{X})^{-1}\mathbf{X}^T =: \mathbf{M}_{\mathrm{POD}} \tag{3.15}$$

Doing this, does not lead to convergence as it immediately solves for all the included POMs but cannot account for those that are not included in the basis. It is suggested to instead use a combined preconditioner:

$$\mathbf{M}_{combo} = \mathbf{M}_{IC}(I_d - \mathbf{A}\mathbf{M}_{\mathrm{POD}}) + \mathbf{M}_{\mathrm{POD}} \tag{3.16}$$

where $\mathbf{M}_{IC}$ is the Incomplete Cholesky preconditioner and I_d the $N \times N$ identity matrix. Using this method the solution does then converge as the components that are not handled by the POD preconditioner are handled by the standard IC preconditioner. For a more detailed analysis see Jiang [2014].

Multiple Preconditioners

The above combined preconditioner is much more costly than a simple IC preconditioner as it requires full $N \times N$ matrix multiplies each time it is used or requires the storage of this large (not necessarily sparse) matrix. Therefore, we instead considered applying the POD preconditioner only once at the start of each linear pressure solve and then revert to a standard

Incomplete Cholesky preconditioner inside the CG loop. This was motivated by the fact that POD preconditioner instantly solves for the POMs [Jiang 2014], and so we proposed this rendered the later inclusion of the POD term in the preconditioner unnecessary. To that end, we also considered the similar case of using the previously unsuccessful POD-only preconditioner (Equation 3.15), and then switching to an IC preconditioner, as it is requires far less computation than Equation 3.16.

Results

Table 3.2 is a breakdown of some tests for the use of a multiple preconditioner scheme for both the POD-only and the combined preconditioner compared to use of a normal Incomplete Cholesky preconditioner. The test simulations were used to reconstruct our input simulation (Figure 3.1) with these schemes, and then test the most successful of these with additional changes to parameters. The results show that when using the same parameters as the input simulation, the combination of POD-only preconditioner and IC does offer a reasonable speed increase, suggesting that the POMs in the basis are quickly resolved. However, the extra computation required for the combined preconditioner, even for a single step, is not worthwhile as the change in iterations is far outweighed by the cost of building the more advanced preconditioner. Then, when considering parameter change, the use of multiple preconditioners has a negative effect on efficiency, causing the solution to require more iterations than using only IC, suggesting the initial application of the POD preconditioner actually diverts the solution. From these results we easily see that this approach does not offer the flexibility we desire.

We now describe some other approaches that may be able to improve the results of these reduced pressure projection methods. As our reduced system is defined by the quality of the POMs, we will first look at methods to improve our POMs for greater accuracy and flexibility.

3.4.2 Divergent Basis

From our early tests, we found that our method suffers from the accumulation of divergence. Treuille et al. [2006] showed that our POMs would be divergence-free. Therefore when projecting onto our POMs, as in the direct projection scheme (Equation 3.14), we should always remain divergence-free as our state is simply a linear summation of a collection of divergence-free POMs. However, our results show otherwise. This seems to arise from our numerical tolerance on our pressure solve, in the PCG solve we do not find the exact solution, simply a close approximation to some tolerance thus we have some small allowed divergence in our solution. When applying the projection (Equation 3.14) we can have values in our reduced coefficient vector $\hat{\mathbf{u}}$ of the order of hundreds, therefore our numerical tolerance errors are then largely inflated. If we look at the divergence values of our POMs against their singular value, Figure 3.6, we can see that this error generally accumulates in our later POMs. Therefore we could reduce our number of POMs being used but this then restricts the motions that we can describe. We instead propose, if we need a motion described by a particular POM, that we could simply perform a further pressure projection with the offending POM as if it were a normal velocity state. This should then allow a greater degree of control on the divergence

		Total IC calls	*Iterations*	*Total Precon*	*Each Precon*	*Total Solve Time*	*Speed change*
input sim, $b = 9.81$	IC + IC (input)	61358	123			67.333s	
	MIC	33000	66			46.3s	1.45x
	POD + IC	12634	26	9.921s	0.01984s	25.408s	2.65x
	combo + IC	12138	25	265.77s	0.53154s	286.314s	0.24x
$b = 12$	POD + IC	50572	101	10.236	0.02047s	66.775s	1.00x
$b = 20$	POD + IC	63401	131	9.821s	0.01964s	75.266s	0.89x
source offset $=+5$	POD + IC	65713	131	10.293s	0.02059s	84.365s	0.79x
	combo + IC	63401	126	200.176s	0.40035s	269.619s	0.25x

Table 3.2: Table of times of a Preconditioned Conjugate Gradient Pressure solve using POD and combination preconditioners for initial pass and Incomplete Cholesky in CG loop.

that we see in our simulation. It should be noted that this will break the optimality of the representation that we get from our SVD.

For re-simulation, from the differences in our final states in Figure 3.8, we can see that the differences between the reduced and full methods are those that will not be solved by even using an ideal divergence-free basis, if it can only display behaviours from a single simulation. Therefore, we will also need to consider further methods to add to these bases to have POMs with a wider range of motion.

3.4.3 Basis Interpolation Methods

As the quality of the POD basis effectively governs the quality of the reduced model and the results have show a very limited range of motion, the basis must be improved to work with a wider variety of simulation parameters. Gerszewski et al. [2013] describe a method for adding new basis vectors defined by an artist into the POD basis to allow additional behaviour in the simulation. To retain fluid motion, these extra basis vectors come in the form of divergence-free velocity fields, which could be calculated with curl-noise [Bridson et al. 2007] or some other model reduction technique [De Witt et al. 2012]. This is then incorporated into the basis by a system of operations that remove any overlapping behaviours between the new entry and the previous modes. However, it is not always intuitive to design every new basis vector that is required to allow a wider range of motion in the simulation. An approach of continuing to use previous simulations as inputs could instead try to use multiple bases from different simulations in this manner. Furthermore, there are a few other ways that have been documented for combining multiple simulations to create more flexible POD bases. These generally come from engineering and use some form of interpolation. These include but are not limited to: global Proper Orthogonal Decomposition (GPOD) [Schmit and Glauser 2004]; sensitivity analysis [Hay et al. 2010]; Grassman interpolation [Farhat and Amsallem 2008].

3.4.4 Fluid Re-Simulation using Subspace Condensation

So far work has focused on extending the pressure projection stage of Kim and Delaney [2013] to work with greater parameter variation. However even then some more fundamental problems must be overcome in order to use this data-driven model reduction technique for fluid re-simulation for VFX. Unlike our previous fixed domain fluid simulations, in reality simulations may have the case of a changing domain, particularly for things like moving collision objects in the flow. Kim and Delaney [2013] use Iterative-Orthogonal-Projection [Molemaker et al. 2008] to solve this issue but they note that this method still does not enforce the boundaries exactly and it can allow flow through the boundary. Other approaches from Treuille et al. [2006] use a precomputed set of velocity fields to resolve the flow around boundaries, but these add to the precomputation time. Gerszewski et al. [2013] use the Vortex Panel method to solve the boundaries in a 2-dimensional simulation but do not extend their method to 3D.

In an attempt to solve these issues in a more general way it could be beneficial to look to the exciting recent work by Teng et al. [2015] combining full and subspace simulation for deformable solids that they call *subspace condensation*. This approach borrows from domain decomposition and requires splitting our simulation domain into a subdomain that is well solved by the subspace and a subdomain that requires full dimensional solution as it is undergoing novel motion. The subspace condensation method couples solutions in the subspace and full dimensional subdomains into a single linear system. This approach may allow incorporation of arbitrary collision objects in the flow by surrounding them with band of full resolution cells and coupling this to the reduced method for the remainder of the domain.

Proposed Method

Consider our linear system for the pressure projection:

$$\mathbf{A}\mathbf{p} = \mathbf{d} \tag{3.17}$$

As previously, suppose that we have a basis $\mathbf{X}$ such that $\mathbf{p} = \mathbf{X}\hat{\mathbf{p}}$ where $\hat{\mathbf{p}}$ is a reduced dimensional analogue of $\mathbf{p}$.

Instead of solving Equation 3.9, return to Equation 3.17, now let us divide up the domain into separate areas, the area to be simulated at full resolution defined by the subscript f, and the area that we have information about and can reduce to the subspace defined by the subscript s. Therefore we now decompose the system (Equation 3.17) with the same method as in Teng et al. [2015] into pressures $\mathbf{p}_f$ and $\mathbf{p}_s$ with corresponding divergence values $\mathbf{d}_f$ and $\mathbf{d}_s$, and our $\mathbf{A}$ matrix becomes:

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{A}_{fs} \\ \mathbf{A}_{sf} & \mathbf{A}_{ss} \end{bmatrix}$$

therefore our linear system is:

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{A}_{fs} \\ \mathbf{A}_{sf} & \mathbf{A}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{p}_f \\ \mathbf{p}_s \end{bmatrix} = \begin{bmatrix} \mathbf{d}_f \\ \mathbf{d}_s \end{bmatrix}$$

Now if we project the subspace region into the reduced space with $\mathbf{X}^T$, and perform the substitution of $\mathbf{p}_s \approx \mathbf{X}\hat{\mathbf{p}}_s$ (as in Teng et al. [2015]), we arrive at:

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{A}_{fs}\mathbf{X} \\ \mathbf{X}^T\mathbf{A}_{sf} & \mathbf{A}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{p}_f \\ \hat{\mathbf{p}}_s \end{bmatrix} = \begin{bmatrix} \mathbf{d}_f \\ \hat{\mathbf{d}}_s \end{bmatrix} \tag{3.18}$$

This system gives a coupling between the full and reduced space representations and is called the subspace condensation method.

The linear system is now reduced to dimension $(N_f + r) \times (N_f + r)$ where N_f is the number of full resolution cells and r the number of POMs in the basis. As in Teng et al. [2015], this may be solved with PCG as suggested using a Jacobi preconditioner for the upper diagonal block and preconditioning the lower diagonal block with its explicit inverse.

With the above method it could be possible to allow an arbitrary number of full dimensional cells to be added in new areas when required, coupled with the reduced subdomain. However, there is still the issue of the definition of an effective ‘oracle’ - used for dividing our domain into full and subspace solved subdomains, such that the result solves the full resolution sufficiently. This means that even areas that are included our reduced basis may need simulating at full resolution for smooth coupling. Similar to both Teng et al. [2015] for new deformations and contacts and Ando et al. [2015] for their approach to solid boundaries, we could suppose that areas undergoing new collisions with solid objects be padded with a boundary layer of full resolution cells. However, unlike Ando et al. [2015] who for static collisions still require padding with full resolution cells, if these static collisions have been previously included in our input simulations, the resulting pressures in the POMs should correctly handle collisions at full resolution and therefore we should be able to continue to use the subspace representation where possible.

Furthermore, as we saw in our earlier tests (and found by Ando et al. [2015], shown by their need for a volume correction scheme), using only a pressure basis with the above Galerkin projection method at the pressure projection, without projecting the velocities as in Equation 3.13, gives insufficient results for correcting the divergence and leads to accumulation of error. It remains to be seen whether the coupling with full resolution cells described above, post-requisite Jacobi iterations (as in Ando et al. [2015]) or similar use of a volume correction method [Kim et al. 2007] can facilitate Equation 3.18 as a usable method in fluid simulation.

3.5 Summary

In this chapter we have described an investigation into model reduction for fluid re-simulation. We have shown a novel use of a data-driven model reduction restricted to the pressure projection stage of a fluid simulation and introduced a new simplified projection operator for this stage. Whilst the current model shows some good results for simple simulations, there is a distinct lack of flexibility towards parameter change that restricts this method's practical use for re-simulation, even in the fixed fully simulated domain case that we have tested here. However, we have also described a collection of avenues of investigation if looking to extend this method to allow a greater variety of parameters through subspace interpolation. We have also proposed a method for coupling full and reduced space sub-domains, in an effort to allow better handling of new collision interaction and domain extension. Unfortunately, although this work was originally undertaken with the hope to extend to liquid simulations, this method does not have the flexibility of Ando et al. [2015] to handle constantly changing domains of the liquid free-surface and so is unlikely to be extendable to liquid simulation.

Whilst the iteration upon the liquid simulation components of the effects described in Chapter 2 is known be an issue, there are a collection of other tools that have been developed to work around this. Technologies like Dynamo allow artists to create simulations quickly, and processes like wedging - running multiple simulations of different parameters simultaneously alleviate some of these issues. However, the artistic iteration and development required to generate the less well-defined secondary simulations (that are not necessarily free-surface liquid simulations) is a particular issue that cannot be mitigated with these approaches. We will look at solutions to this in the coming chapters.

Chapter 4

Increasing Detail: Physically-Based Droplet Interaction

The majority of the content of this chapter was presented in the paper ***Physically-Based Droplet Interaction*** *at SCA 2017. The implementation described has been developed as a component of a prospective updated version of Dynamo. However due to production constraints this framework has not yet been exposed to artists at DNEG. There are plans for this to be released into production in the future.*

4.1 Overview

This chapter presents a system for the practical and efficient modelling of physically-plausible droplet interactions on a mesoscopic scale, for use in medium to large scale energetic liquid simulations. The approach taken - to directly model interactions between droplet particles - avoids the requirement of very high resolution to resolve the small physical scale of these droplet effects that would otherwise come with using standard Eulerian or Lagrangian methods. This allows the creation of high quality results for sprays of millions of plausibly interacting droplets that would be very difficult or impractical with existing systems, due to their required increase in resolution, and so, computational complexity, or through the large amount of artistic effort usually required (§2.5).

4.2 Motivation

Small-scale phenomena play an important part in maintaining a high level of visual detail and plausibility in practically every simulated effect used in VFX production from fluids (droplets and sprays); destruction (dust and particulate matter); to character simulation (muscles, wrinkles and fur). These finer details ground large-scale effects in reality, often being more important to believability than their larger scale counterparts, a characteristic that allows even hand-animated approaches for some effects such as the massive waves from the water planet in *Interstellar* [Nolan 2014]. Whilst these phenomena are recognised to be important and are subsequently required for almost all liquid effects, their recreation is often subject to many different approaches in practice [Iversen and Sakaguchi 2004; Froemling et al. 2007]. One common theme amongst most existing methods for these kind of phenomena is that they are almost entirely artistically-driven, often requiring a lot of time and effort to create bespoke setups on a shot-by-shot basis. Taking this approach, artists are required to control all visual characteristics of the effects they wish to create, including the impression of the interactions that occur within them and their resultant behaviours - which can be particularly troublesome for very complex interactions such as those in liquid motion.

Droplets, the small-scale elements that make up liquid splashes and sprays, interact with one-another primarily through direct collisions between their surfaces. These small (approximately spherical) dispersed liquid masses undergo interactions that lead to the deformation of their surfaces, merging with one-another and even fragmentation (break-up into smaller droplets). Such interactions drive the evolution of the size, shape and velocity distributions of droplets and so the sprays [Kim et al. 2009] as a whole. In this way, by recreating these behaviours we could reproduce both localised fine-scale per-droplet interactions and the larger-scale influences and motions of liquid sprays.

However, as we have seen in §2.4, due to the small-scale and multi-phase nature of droplet interactions, these behaviours can be difficult to model, especially using the often simplified simulation methods favoured in computer graphics [Froemling et al. 2007; Ihmsen et al. 2012]. As such, these behaviours (and often even visual representation of any such small-scale liquid phenomena) have been largely ignored and neglected by computer graphics researchers and artists alike. Instead artists are often left to attempt to recreate the resulting larger scale behaviours of sprays directly using bespoke and non-physical methods, with great difficulty. In this chapter, we consider how to efficiently recreate these interactions or more specifically, their resulting phenomena and the processes behind them.

The decoupled droplet and spray methods currently used in production take the form of a variety of techniques, capturing varying levels of quality and detail/interaction (Chapter 2). A common approach is to use secondary particle simulations: either simple particle systems [Ihmsen et al. 2012]; non-standard use of a particle-based liquid simulation frameworks, or some bespoke system that lies somewhere in between [Losasso et al. 2008; Chentanez et al. 2015a]. We will now examine these existing approaches before describing our own later in the chapter.

4.3 Related Work

The modelling of droplet and spray interactions in computer graphics has been the subject of a few distinct approaches, many of which we introduced in the Chapter 2. In this section we will briefly expand on our review of these works with particular focus on their approaches to modelling the interaction between droplets within splashes/sprays.

Particle approaches: The most popular approaches for liquid simulation in computer graphics, and as such the most widely used techniques to incorporate simulation of droplets, are particle-based approaches. Apart from the workflow advantages of using particles (mentioned in §2.3), they also offer geometric advantages in droplet simulation, most notably the visual similarity between a spherical particle and a droplet of liquid. In this way, by modelling droplets and sprays with particles we are able to arrive at a renderable set of droplets, simply considering the simulated particle set itself.

Use of particle-based methods for droplet simulation has been documented by many authors in pursuit of high fidelity splash and spray recreation [Takahashi et al. 2003; Song et al. 2005; Kim et al. 2006; Losasso et al. 2008; Mihalef et al. 2009; Ihmsen et al. 2012; Yang et al. 2014; Chentanez et al. 2015a]. The most well-known and widely used of these methods (including those in DCCs such as Houdini [Side Effects 2018]) work by processing an initial liquid simulation and from this creating secondary systems of particles to represent droplets and fine-scale liquid elements that would otherwise be missing using standard liquid simulation techniques. Many of these methods consider only ballistic particles, with interesting behaviour driven by external influences [Takahashi et al. 2003; Song et al. 2005; Kim et al. 2006; Mihalef et al. 2009; Ihmsen et al. 2012], and, as they lack internal interaction, are only in practice suitable for scenarios with large scale differences i.e. large oceans and very small droplets/fine spray or splashes of very high resolution free-surface simulations that require similarly fine spray elements. Other methods consider a combination of liquid simulation techniques to try to incorporate different scales of motion and overcome the issues that arise with this lack of internal interaction. These include: the use of a modified SPH/FLIP/PIC hybrid allowing the targeting of differing particle densities and blending to ballistic motion for low density particles from Losasso et al. [2008]; Yang et al. [2014] and the use of single iterations of Lagrangian particle methods such as PBD [Macklin and Müller 2013] or SPH [Ihmsen et al. 2014b] velocity updates alongside an otherwise FLIP/PIC-based scheme of Chentanez et al. [2015a]. These methods look to create interaction between the particles such that they can be used in simulations requiring visible interaction between droplets, whilst avoiding increased resolution requirements of the base simulations. However the models they appropriate do not capture realistic droplet-like behaviours (§4.3.1).

Further distinctions can be made between droplet particle models that are required to be simulated at the time of the free-surface simulation, such as Takahashi et al. [2003]; Song et al. [2005]; Kim et al. [2006]; Yang et al. [2014]; Chentanez et al. [2015a] or those that can be run as a post-process [Mihalef et al. 2009; Ihmsen et al. 2012]. Whilst the latter do not allow two-way coupling or mass-preservation in droplets, in our experience we have seen that the decoupled approach is preferred in production, allowing an artist to break up the tasks of finalising the free-surface liquid simulation and the finer elements that are required on top such as those of spray, droplets, foams and mist.

Volumetric approaches: Taking a volumetric approach to spray simulation, Nielsen and Østerby [2013] develop a physically-based model for using a two-continua (air/liquid) discretisation. Motivated as a more physically-accurate representation of spray than say, ballistic particle methods, their method considers a macroscopic, volumetric approximation with a focus on motion with respect to the surrounding simulated air and indirect interaction, recreating larger multi-phase interactions rather than individual visible droplet details. In more simplified volumetric approaches, Takahashi et al. [2003] and Yang et al. [2014] consider transitions of droplets to finer scale mist by using accompanying density volumes that are advected through a rasterised droplet velocity field and a simple single-phase gas simulation respectively.

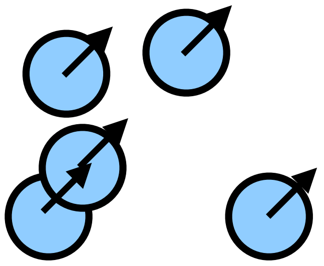
Other Eulerian approaches to droplets have instead considered recreation of more microscopic phenomena such as the work on droplet and liquid flow along solid objects using signed distance functions from Wang et al. [2005].

Mesh approaches: Finally, simulation of small-scale liquid phenomena including droplet interactions have been exhibited using the model of Da et al. [2016], who similarly consider surface-tension dominated liquid surface phenomena as we do in this chapter. However, their work has a very different focus of high quality tracking of deforming liquid surfaces, at smaller scales. We instead approximate this behaviour using a simple particle-based method that allows application to systems of millions of droplets with relative ease, as well as integration into existing production simulation workflows.

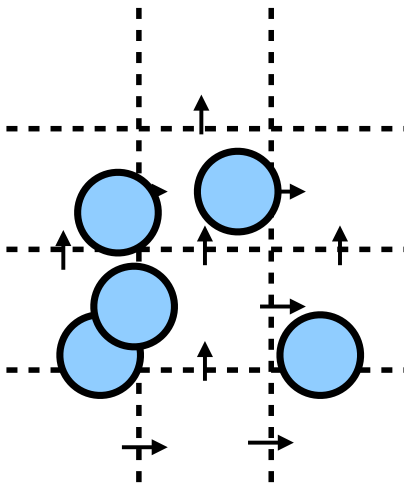
4.3.1 Issues with Existing Droplet Methods

When using particle-based methods for droplets and sprays, a common approach to rendering the final result is to render the simulated particles directly, simply as spheres or disks. However, to properly consider the suitability of the above interaction methods, it is important to look at the physical properties of these methods and the phenomena they are able to recreate on the particle-as-droplet scale. For example, in all standard particle-based approaches to liquid simulation (FLIP/PIC/SPH/PBF), whilst able to carry physical quantities such as mass or density, particles are ultimately used as ways of tracking the collective fluid mass and are not explicitly representative of, or in fact act (in isolation) as, a real fluid mass such as a droplet. In this way, considering these simulated particles as representative of real liquid droplets (implied by rendering them directly) is inappropriate as the real life phenomena that they represent are subject to very different interactions than those that they are able to describe, e.g. droplet breakup and merging (§4.4) vs repulsion and bouncing. Thus, when isolated or sparsely sampled, many of these are possibly more representative of say marbles than water droplets.

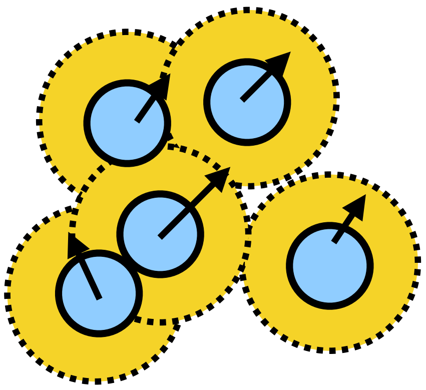
Consider first the hybrid grid-particle liquid simulation framework (PIC/FLIP/APIC) as described in Chapter 2. In this method, introduced to graphics by Zhu and Bridson [2005], particles are simply an advection mechanism, that is, a method to transport fluid quantities throughout the simulation, transferring to and from a background grid each timestep in order to calculate interaction. Here the actual liquid representation is grid-based, described by the velocity field and often a level-set surface. Most notably, the actual internal interaction (through pressure) is solved on the grid discretisation, not the particles. Interactions that are captured by these methods are therefore driven by, and occur on the scale of, the grid resolution. This separation of interaction and advection simulation elements results in particles of secondary importance to the simulation, not typically required to be preserved and/or mass carrying. One advantage of this means they are able to be trivially resampled under a representative surface (if it exists) - to improve the accuracy of the motion tracking without largely effecting the overall motion resolution. However, such a separation of simulation elements and their different roles can be troublesome for less well-defined scenarios such as isolated particles and areas that become sparsely sampled as they neither accurately capture larger scale motions nor motions on a small enough scale to meaningfully represent droplets.



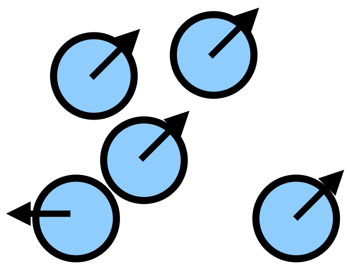
(a) **Ballistic:**
Particles are completely void of interaction with one-another relying entirely on (larger-scale) external interaction to cause visible changes to their motion.



(b) **Hybrid (e.g. FLIP/PIC):**
Particles stamp properties to a background grid to resolve interaction, reducing the fidelity of the interaction and amount of detail captured to that of the grid resolution.



(c) **SPH/PBF:**
Particles are effectively radial kernels, attracting and repelling each other within their sphere of influence dependent on a collection of forces.



(d) **Rigid Body:**
Particles move and interact more akin to marbles than liquid droplets, bouncing off each other being their only interaction.

Figure 4.1: Standard particle-based droplet simulation techniques and their interaction mechanisms. In no existing method does a single particle meaningfully represent a mass of liquid such as a droplet.

Similarly, consider an SPH simulation. Whilst particles are mass carrying, they are a mechanism to evaluate an approximation of the integral of the fluid equations throughout space. In this way, areas of low particle density/reduced neighbourhood offer a poor approximation for these equations (optimal neighbourhood being 50 particles [Winchenbach et al. 2016]). In fact areas of low density are a known issue in SPH simulations (unless additional air particles are also used), exhibiting what is known as the tensile instability [Macklin and Müller 2013] as particles clump together due to artificial tension forces. Similarly it has been found that even with multi-phase simulation setups isolated liquid particles do not act in a droplet-like manner [Szewc et al. 2016].

Given these kind of issues, it is clear that if using FLIP, SPH or other similar standard liquid simulation frameworks to drive the motion in droplets we fall foul of rendering simulation elements rather than particles representative of droplets in reality. Whilst these particle methods offer a flexible and relatively intuitive interface for artists and users of the systems for larger scale liquid simulations, the final result of such approaches should always be a renderable liquid surface (the free-surface), created (often as a post process) from the simulated particles altogether (see Chapter 5). The collective motion of the particles give plausible behaviours for larger bodies of liquid as they move and deform, but the particles themselves are not the representation of water droplets that we seek.

Other approaches have seen non-liquid alternative interaction schemes for droplet particles such as rigid body motion [Tomar et al. 2010] and even purely ballistic motions, forgoing interaction altogether. These schemes can be readily seen as insufficient for creating droplet interactions either immediately (ballistic, for obvious reasons) or as they do not consider important liquid properties such as deformable and merging surfaces.

From these observations we can see why previous approaches, seemingly borne out of familiarity more than suitability, do not fulfil the requirements for a system of high quality droplet simulation and are unable to capture the behaviours that occur in reality on these small scales (such as those shown in Figure 4.2). The approach we take going forward is to instead consider a closer look at the interactions occurring between droplets in reality, the mechanisms behind them and then consider how we may recreate them. Only then can we look to develop a system able to use droplet particles to recreate plausible droplet and spray phenomena.

4.4 Real Droplet Interactions

Water droplets are (approximately) spherical masses of liquid submersed in air, with surface tension forces creating their spherical shape, undergoing interactions with the surrounding air through drag, and acting under the force of gravity. These droplets are often created as masses of liquid break away from a main body either, as a collection of larger isolated droplets, a finer spray or a combination of the two. On this scale, and with these kind of surface area-volume ratios, surface tension forces dominate and are the deciding factor on how droplets interact with one-another. Considering droplets in these collections/sprays as separate masses, we look at the interactions that occur as droplet surfaces come into contact and the results thereof, hereafter known as *droplet collisions*.

Within the engineering community, interactions between colliding droplets have been studied through observations of those that occur between binary pairs of droplets[1]. In our work, we use these observations to inform our description of the range of internal interactions that occur between droplets in sprays.

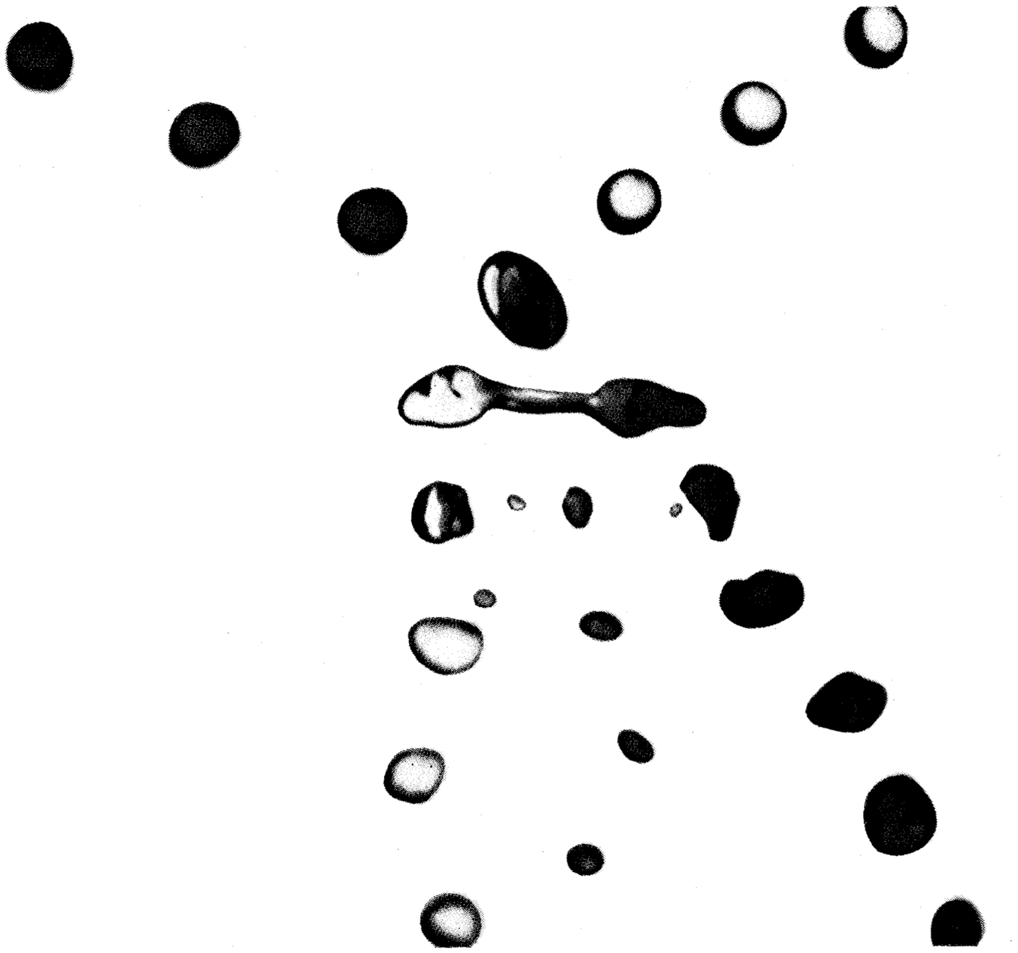


Figure 4.2: Droplet collision from experiment: initial coalescence of interacting droplets followed by subsequent separation and fragmentation. Reproduced with permission from Brazier-Smith et al. [1972].

4.4.1 Binary Droplet Collisions

There are a number of distinct outcomes that occur from binary collisions between liquid droplets. These can be broken down into the following outcomes: bounce, coalescence,

[1]As the likelihood of multiple collisions occurring at the *exact* same time is unlikely, these observations are able to offer meaningful insight into these systems. Therefore, from now on we discuss binary collisions with a view to explaining larger-scale behaviours as a consequence.

stretching separation, reflexive separation (and shattering) [Qian and Law 1997; Orme 1997; Estrade et al. 1999; Pan et al. 2009].

These are defined as follows:

- *Bounce* - droplets that move towards each other with enough energy such that there is insufficient time for ambient surrounding air to escape from in-between, but not enough to displace it, seemingly bounce off of one another. As the air between them compresses, it creates a pressure forcing the droplets away from one another, causing deformation of their surfaces and the appearance of bouncing apart. As there is no direct interaction of the surfaces due to the compressed air pocket, droplet masses are unchanged.

- *Coalescence* (Figure 4.3 - left): as droplets come into contact, the tension from their resulting combined surface causes them to merge to form a single mass of liquid. The free surface of the combined mass initially oscillates and contracts due to the tension forces on their now deformed surface, working against any incident opposing kinetic energy. These oscillations dissipate as the combined mass tends to a final (approximately) spherical stable droplet state.

- *Stretching Separation* (Figure 4.3 - middle): grazing collisions, where droplets move past each other with large relative velocity and low surface overlap, instead of fully coalescing, form a temporary fluid ligament that connects them. The majority of the droplet masses continue along their initial trajectories, causing the ligament to stretch and subsequently break. As the tension on the broken ligament causes it to contract, its mass either rejoins the initial droplets or creates a collection of smaller, satellite droplets.

- *Reflexive Separation* (Figure 4.3 - right): initially coalescing collisions with sufficient kinetic energy to cause extreme initial deformation and oscillations that subsequently break-up the combined mass and split into a collection of smaller satellite droplets. The point in the oscillation at which break up occurs varies dependent on the incident kinetic energy. For less extreme velocity differences, and so lesser initial deformation, this stretching and fragmentation occurs along the relative trajectory of the colliding droplets in the prolate phase of oscillation, but for increased initial kinetic energy, more aggressive, *shattering* behaviour occurs instead [Pan et al. 2009] in the oblate phase perpendicular to the incident droplet velocities.

4.5 Droplet Collision Modelling

Droplet collisions and their resulting behaviours are the main interactions that occur within liquid sprays and yet such phenomena have until now been largely ignored by the graphics community[2]. Introducing a mechanism to recreate these behaviours into a ballistic particle

[2]Nielsen and Østerby [2013] and Mihalef et al. [2009] recognise the existence of these phenomena, but instead consider these interactions at a volumetric, more macroscopic scale which fails to capture them at high visual fidelity.

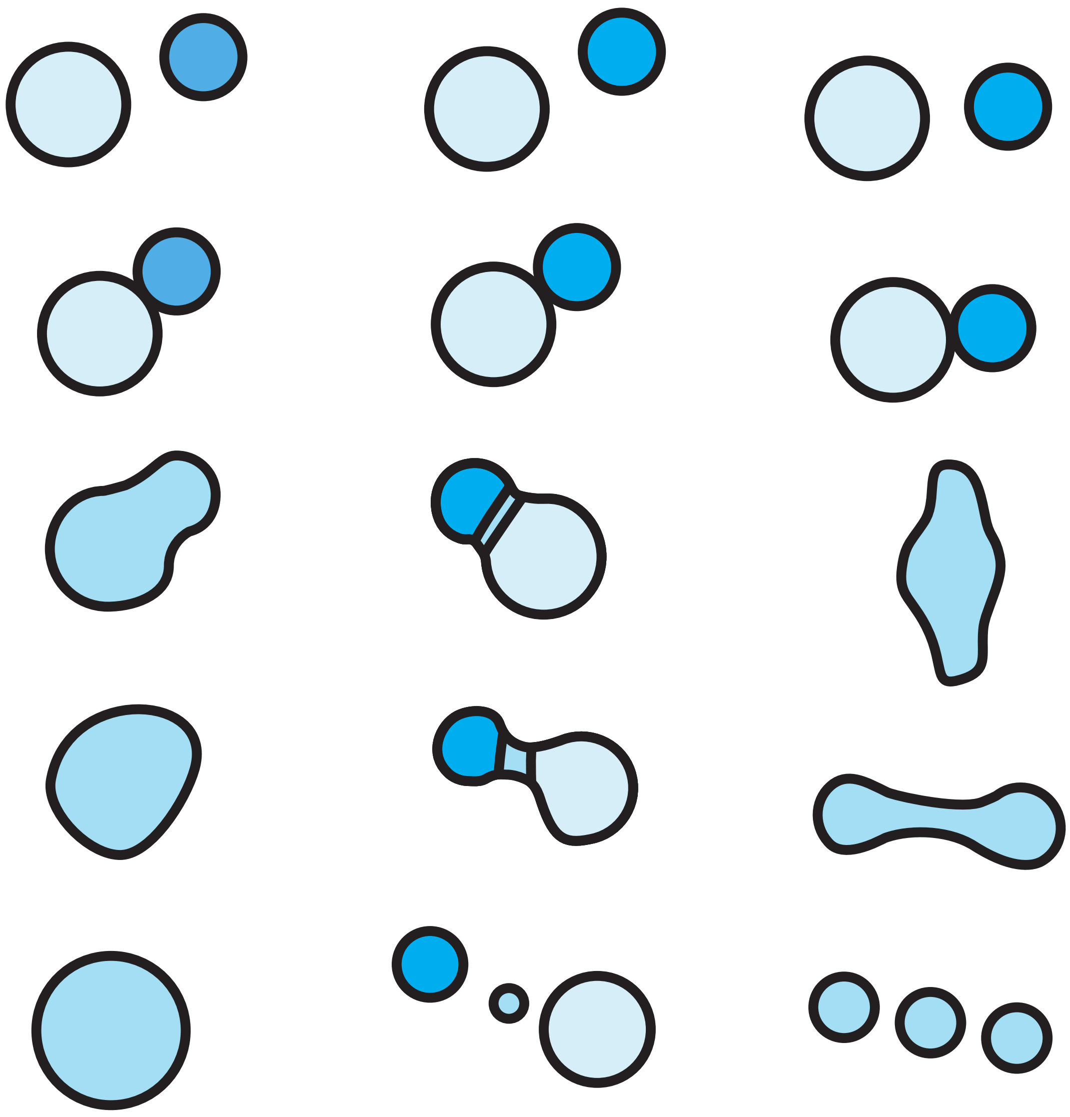


Figure 4.3: Main droplet collision types (start - top, end - bottom); Left to right - **Coalescence**, **Stretching Separation** and **Reflexive Separation** (prolate phase).

system allows the particles to more meaningfully represent droplets[3] and so recreate small-scale liquid details (as well as drive larger-scale evolution of particle distributions). Modelling of droplet collision interactions in this way is well-studied and documented in the fields of engineering and meteorology, where it is used to alleviate requirements for high resolution multi-phase simulation to simulate interacting liquid sprays [O'Rourke and Bracco 1980; Kim et al. 2009]. Given the collection of common behaviours that have been observed in reality (§4.4.1), an approach often used in these fields is to attempt to parametrise, predict and directly recreate these observed outcomes (or a subset thereof), e.g. those of coalescence and separation (Figure 4.3), with the use of discrete threshold methods [Brazier-Smith et al. 1972; Ashgriz and Poo 1990]. These methods have been shown to accurately and efficiently model interactions between large numbers of droplet particles and as such open the door to creating more meaningful recreations of spray phenomena at high fidelity. By considering specific behaviours in this way, we can allow the incorporation of phenomena unique to droplet interactions such as higher energy collisions causing fragmentation and breakup of droplets [Munnannur and Reitz 2007]. This behaviour for example, acts as a mechanism for droplet particle emission to occur naturally throughout our simulations and helps add finer-scale detail into the system. For the remainder of this chapter, we look to introduce these methods and their unique behaviours into the computer graphics tool-set.

Combining physically-validated models with novel extensions to accommodate their use in graphics, we develop a method to simulate droplets and their interactions on a mesoscopic scale whereby a single droplet particle is the smallest interacting element yet remains able to reproduce realistic behaviours on this scale. This method is shown to be to able produce high quality results for millions of small, yet visible, droplets in a spray and so can facilitate the production of more plausible results than is possible with existing methods.

By the discrete nature of these droplet collision models, most existing contributions in the field break these models down into two stages; determining the collision outcome that should occur; and the way this outcome manifests in post-collision characteristics of the droplets. Our work is no exception to this. To give context to the decisions we have made in development of our model, we will briefly summarise the existing works that have led to the development of this approach, then describe the choices and extensions we make for our own model. We begin by looking at the way that these approaches quantify a droplet collision.

4.5.1 Parametrising Droplet Collisions

From now on consider the collision between droplets i and j with radii $r_i \geq r_j$, velocities $\mathbf{u}_i, \mathbf{u}_j$ and positions $\mathbf{x}_i, \mathbf{x}_j$. Following the methodology outlined in the seminal work of Brazier-Smith et al. [1972], a collision is described in terms of the intrinsic collision parameters We, X, δ where:

- **Weber Number**, We: The ratio of inertial forces to surface tension forces for a given fluid

[3]In computer graphics, the scale of a 'droplet' is usually somewhat different to that in engineering references ($O(\mu\text{m})$) but in this paper, we use the realistic collision behaviours of droplets, and their motivating physical interactions, as an approximation of the interactions that should occur between our often larger liquid 'droplets'. We consider droplets to be any visible isolated form of water that can be reasonably approximated by a sphere.

element. It is given by:

$$We = \frac{\rho \Delta \mathbf{U}^2 L}{\sigma} \tag{4.1}$$

for characteristic length L, fluid material density ρ, relative velocity $\Delta\mathbf{U}$ and surface tension coefficient σ. For our scenario of two droplets colliding, the characteristic length L is diameter of the smaller droplet $d_j = 2r_j$ and the relative velocity is given by $\mathbf{u}_{ij} = \mathbf{u}_j - \mathbf{u}_i$ s.t. we have:

$$We_{ij} = \frac{2\rho r_j ||\mathbf{u}_{ij}||^2}{\sigma} \tag{4.2}$$

The Weber number is a quantity often used in engineering applications (and sometimes considered in graphics applications e.g. Mihalef et al. [2009]) in analysis of the properties of interface between two flows (e.g liquid and air).

- **Impact**, X: The shortest distance between the two droplet centres orthogonal to the relative velocity (Figure 4.4), normalised by the droplet radii gives the impact parameter:

$$X = \frac{x}{r_i + r_j} = \frac{||\mathbf{x}_{ij} - \mathbf{x}_{ij}^T \hat{\mathbf{u}}\hat{\mathbf{u}}||}{r_i + r_j} \tag{4.3}$$

where x is the perpendicular distance between droplet i and the line along $\mathbf{u}_{ij}$ from $\mathbf{x}_j$, $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ and $\hat{\mathbf{u}} = \frac{\mathbf{u}_{ij}}{||\mathbf{u}_{ij}||}$.

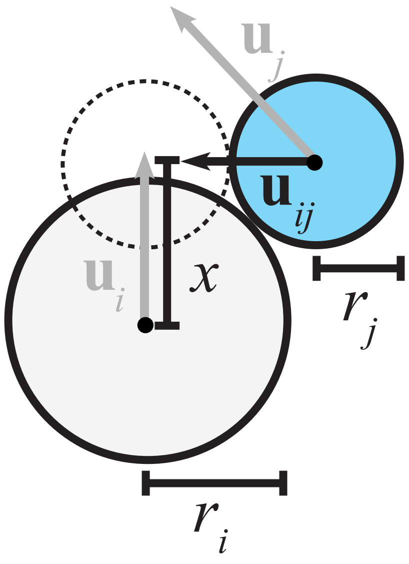


Figure 4.4: Geometric collision parameters for colliding droplets i and j.

- **Size Ratio**, δ: The ratio of the smaller droplet radius to the larger droplet radius:

$$\delta = \frac{r_j}{r_i} \tag{4.4}$$

These parameters define key geometric, physical and dynamic properties of the collision between a pair droplets[4] and can be used in both the prediction of an appropriate collision outcome and the resulting post-collision characteristics.

4.5.2 Collision Outcome Classification

Given the well-defined set of outcomes that occur from binary droplet collisions, a collection of models have been developed that attempt to explain these outcomes and, more importantly, describe how to predict which should occur for a given droplet collision.

Brazier-Smith et al. [1972]'s work on rain droplets first introduced the notion of direct droplet collision modelling. Their model described the phenomena behind coalescence and stretching separation collisions by defining a threshold between them using the collision intrinsics We, X and δ. The criterion proposed for separation was for the rotational energy about the centre of mass to be greater than the surface energy required to reform the initial two droplets from the nominal combined droplet with radius $r_0 = \sqrt[3]{r_i^3 + r_j^3}$. Analysis of these energies leads to the following threshold on the impact parameter X:

$X > X_{coalescence} \rightarrow$ *stretching separation* for $X_{coalescence}$ s.t.

$$X_{coalescence} = \sqrt{e_{coalescence}} \tag{4.5}$$

for

$$e_{coalescence} = \frac{2.4}{We} f\left(\frac{1}{\delta}\right) \tag{4.6}$$

and $f(\cdot)$ s.t.

$$f(\gamma) = \frac{1 + \gamma^2 - (1+\gamma^3)^{\frac{2}{3}}(1+\gamma^3)^{\frac{11}{3}}}{\gamma^6(1+\gamma)^2} \tag{4.7}$$

This threshold was then introduced into simulation by O'Rourke [1981] when the above coalescence/separation threshold was applied to spray modelling, approximating the function $f(\cdot)$ with the polynomial

$$f_{O'Rourke}(\gamma) = \gamma^3 - 2.4\gamma^2 + 2.7\gamma \tag{4.8}$$

Following this, Ashgriz and Poo [1990] developed an alternative method defining two thresholds, one as an improvement on the threshold between coalescence and stretching separation and the other between coalescence and reflexive separation. These thresholds are defined using a collection of kinetic and surface energy balance equations. This work instead considers the kinetic energies of opposing components of droplet spheres (for both stretching and reflexive separations) rather than the angular momenta of Brazier-Smith et al. [1972]. This alternative formulation, better matching their experimental results, is motivated by an experimental observation that stretching separation occurs before the development of any significant angular motion. The thresholds of this method are instead formulated in terms of the Weber number We as:

[4]An important property of these parameters is their non-dimensionality, as such these models are generally applicable to a wide range of droplet sizes as long as the other assumptions of any model using them are justified.

We $>$ *We*$_{stretching}$ $\rightarrow$ *stretching separation* for *We*$_{stretching}$ s.t.

$$We_{stretching} = \frac{4(1+\delta^3)^2[3(1+\delta)(1-X)(\delta^3\phi_j+\phi_i)]^{\frac{1}{2}}}{\delta^2[(1+\delta^3)-(1-X^2)(\phi_j+\delta^3\phi_i)]} \tag{4.9}$$

where ϕ_i, ϕ_j (Figure 4.5) are the fractions of the droplet volumes V_i, V_j for i, j resp. that interact with the other droplet during the collision, given by

$$\phi_i = \begin{cases} 1-\frac{(2-\tau)^2(1+\tau)}{4}, & \text{if } h > r_i. \\ \frac{\tau^2(3-\tau)}{4}, & \text{otherwise.} \end{cases} \tag{4.10}$$

$$\phi_j = \begin{cases} 1-\frac{(2\delta-\tau)^2(\delta+\tau)}{4\delta^3}, & \text{if } h > r_j. \\ \frac{\tau^2(3\delta-\tau)}{4\delta^3}, & \text{otherwise.} \end{cases} \tag{4.11}$$

and

We $>$ *We*$_{reflexive}$ $\rightarrow$ *reflexive separation* for *We*$_{reflexive}$ s.t.

$$We_{reflex} = \frac{3[7(1+\delta^3)^{\frac{2}{3}}-4(1+\delta^2)]\delta(1+\delta^3)^2}{(\delta^6\eta_1+\eta_2)} \tag{4.12}$$

where $\eta_1 = 2(1-\xi)^2(1-\xi^2)^{1/2}-1, \eta_2 = 2(\delta-\xi)^2(\delta^2-\xi^2)^{1/2}-\delta^3$ for $\xi = 0.5X(1+\delta)$. Further description of the derivation of these thresholds can be found in the Appendix.

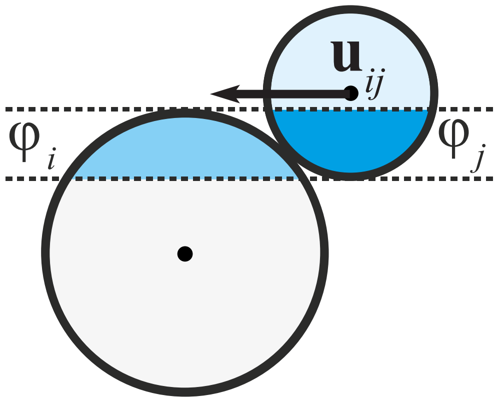


Figure 4.5: Interaction volumes of droplets i and j. ϕ_i, ϕ_j are the fractions of the droplet volumes that are overlapped as droplet j is swept along the velocity $\mathbf{u}_{ij}$.

Estrade et al. [1999] define a threshold for bouncing outcomes from collisions. The condition for bouncing is that the kinetic energy in the collision should not surpass the energy required to produce some limit deformation in the colliding droplets. Their result is given by a threshold on *We*:

We $<$ *We*$_{bounce}$ $\rightarrow$ *bounce* for *We*$_{bounce}$ s.t.

$$We_{bounce} = \frac{\delta(1+\delta^2)(4\Theta-12)}{\phi_i(\cos(\arcsin(X)))^2} \tag{4.13}$$

where the shape factor $\Theta = 3.351$.

These models have also been combined to capture a range of droplet behaviours in later works [Munnannur and Reitz 2007]. However, to our knowledge no similar criterion has been developed to account for the transition from reflexive separation to *shattering* apart from simple constant *We* thresholds of 100 and 1000 in Georjon and Reitz [1999], although some analysis of this behaviour has been undertaken [Pan et al. 2009].

4.5.3 Post-Collision Characteristics

Whilst determination of the outcome type is important, methods to calculate meaningful post-collision characteristics are similarly required so that it is possible to then represent these outcomes in a simulated system. The effects of these droplet collision interactions manifest in three main ways: changes to the topology of the droplets (that is, merging and splitting), redistribution of liquid mass between droplets, and, updates to the velocities and trajectories of the droplets.

Coalescence

For coalescence, we can assume simple mass and momentum conservation, combining the incident droplets i,j into a single droplet k s.t.

$$m_k = m_i + m_j \tag{4.14}$$

$$r_k = \sqrt[3]{r_i^3 + r_j^3} \tag{4.15}$$

$$\mathbf{u}_k = \frac{m_i}{m_k}\mathbf{u}_i + \frac{m_j}{m_k}\mathbf{u}_j \tag{4.16}$$

and the position of the resulting droplet post collision is therefore given by

$$\mathbf{x}_k = \frac{m_i}{m_k}\mathbf{x}_i + \frac{m_j}{m_k}\mathbf{x}_j \tag{4.17}$$

for m_i, m_j, m_k the mass of the incident and combined droplets respectively.

Separation

For stretching separation, the work of O'Rourke [1981] suggests a method to calculate the post-collision velocities as each droplet perturbs the other's motion, this velocity update is given by:

$$\tilde{\mathbf{u}}_k = \frac{r_k^3\mathbf{u}_k + r_l^3\mathbf{u}_l - r_l^3(\mathbf{u}_k - \mathbf{u}_l)z}{r_k^3 + r_l^3} \tag{4.18}$$

for $k, l = i, j$ where

$$z_{stretching} = \frac{X - X_{coalescence}}{1 - X_{coalescence}} \tag{4.19}$$

Tennison et al. [1998] then proposed a similar formulation in their analysis of reflexive separation, replacing z with

$$z_{reflexive} = \sqrt{1 - \frac{\text{We}_{reflex}}{\text{We}}} \tag{4.20}$$

This formulation for velocity update can also be used to calculate the post-collision velocity of bouncing by decomposing the velocities down into relative and normal components [Kim et al. 2009] i.e. $\mathbf{u}_{i,rel} = \mathbf{u}_i \cdot \hat{\mathbf{u}}_{i,j}$ and $\mathbf{u}_{i,norm} = \mathbf{u}_i - \mathbf{u}_{i,rel}$ and applying the update (Equation 4.18) to only the relative velocity components, where $-z$ is instead the coefficient of restitution (i.e. $z = -1$, perfectly elastic collision). Reforming the velocities $\tilde{\mathbf{u}}_k = \mathbf{u}_{k,rel} + \mathbf{u}_{k,norm}$ gives post-bounce velocities. Kim et al. [2009] further propose the use of a similar decomposition of the velocity in the stretching separation update, such that only the relative velocity component is updated by Equation (4.18).

Fragmentation

Whilst velocity updates are an important part of the separating phenomena, there is also the possibility of fragmentation (i.e. splitting into satellite droplets) that can occur during these outcomes. There have been two previous approaches that look to describe this phenomenon for these threshold models, those of Ko and Ryou [2005] and Munnannur and Reitz [2007].

Ko and Ryou [2005] extend the model of Ashgriz and Poo [1990] to account for fragmentation. That is, they calculate the number of satellites from the energy difference between the incident and predicted post-collision states of the droplets. In doing so, they describe a method for calculating the volume of the incident droplets that is included in the temporary ligament and so contributes to satellite droplets. For stretching separation, following observations made by Ashgriz and Poo [1990] and Qian and Law [1997] this is given by some fraction of the overlapping volumes based on the comparative energies present in the collision, and for reflexive separation by the entire incident droplet volume. This calculation is also used (with slight modification) by Munnannur and Reitz [2007] in their alternative approach to fragmentation.

In their method, Munnannur and Reitz [2007] combine components from O'Rourke [1981], Ashgriz and Poo [1990], Estrade et al. [1999], Georjon and Reitz [1999] and Ko and Ryou [2005] to model a wide range of behaviour, including fragmentation. Unlike Ko and Ryou [2005] however, they instead focus on explaining fragmentation by considering the physical mechanism behind the ligament formation and its breakup into satellites. They consider fragmentation during collision to be the result of the break-up of a temporarily-formed cylindrical liquid ligament due to the Plateau-Rayleigh instability [Rayleigh 1878], modelling its stretching and eventual collapse/retraction. Given the interaction volume (as in Ko and Ryou [2005]) and the stretching velocity $\mathbf{u}_{ij}$, an equation for the cylindrical radius at breakup of the ligament can be calculated, and in turn, the radius of any resulting satellite droplets [Georjon and Reitz 1999]. This can be calculated as follows.

Ligament volume: For stretching separation, the liquid ligament forms between the two droplets as they graze past one-another. This volume, V_{lig}, is assumed to be equal to some fraction of the total interaction volume of the droplets, dependent on the magnitude of the opposing surface, stretching and viscous dissipation energies (as used by Ashgriz and Poo [1990] to calculate the collision type thresholds) in the collision s.t.:

$$V_{lig} = C_{sep}(V_{interact,i} + V_{interact,j}) \tag{4.21}$$

where $V_k = \frac{4}{3}\pi r_k^3$ and $V_{interact,k} = V_k\phi_k$ for $k = i, j$ and ϕ_k as in Figure (4.5) and $C_{sep} < 1$, known as the separation volume coefficient, is

$$C_{sep} = \frac{\mathbf{E}_{stretch} - \mathbf{E}_{surface} - \mathbf{E}_{dissip}}{\mathbf{E}_{stretch} + \mathbf{E}_{surface} + \mathbf{E}_{dissip}} \tag{4.22}$$

where

$$\mathbf{E}_{stretch} = \frac{1}{2}\rho||\mathbf{u}_{ij}||^2 V_i\left(\frac{\delta^3}{(1+\delta^3)^2}\right)[(1+\delta^3)-(1-X^2)(\phi_j+\delta^3\phi_i)] \tag{4.23}$$

$$\mathbf{E}_{surface} = 2\sigma[\pi V_i r_i \tau(\phi_i + \delta^3\phi_j)]^{\frac{1}{2}} \tag{4.24}$$

for $\tau = (1-X)(1+\delta)$ and $\mathbf{E}_{dissip}$ is 30% of the initial kinetic energy of the colliding droplets.

For reflexive separations, this ligament is considered to be the entire temporarily merged mass s.t. $V_{lig} = V_i + V_j$, which is instead stretching due to internally-driven deformations along the relative velocity (known as prolate phase oscillation).

Ligament instability: The radius at breakup r_{bu} of the cylindrical liquid ligament (assuming initial length equal to its radius r_0) extending with velocity $\mathbf{u}_{ij}$, is then given as the solution to:

$$\beta \mathrm{We}_0^{\frac{1}{2}}\left(\frac{r_{bu}}{r_0}\right)^{\frac{7}{2}} + \left(\frac{r_{bu}}{r_0}\right)^2 - 1 = 0 \tag{4.25}$$

where $\beta = \frac{3}{4\sqrt{2}}(k_1k_2)$, $k_1 = 11.5, k_2 = 0.45$ (using updated constants from Kim et al. [2009]) and $\mathrm{We}_0 = 2r_0\frac{\rho}{\sigma}||\mathbf{u}_{ij}||^2$.

Finally, assuming that this breakup occurs due to the disturbance of the ligament that has the maximum growth rate and that the volume of each satellite is equal to the cylinder of length corresponding to the wavelength of this disturbance, Munnannur and Reitz [2007] (citing Georjon and Reitz [1999]) give the radius of each satellite as $r_{sat} = 1.89r_{bu}$. Using r_{sat} and the ligament volume V_{lig} the number of satellites can be calculated using mass conservation:

$$n_{sat} = \frac{V_{lig}}{\frac{4}{3}\pi r_{sat}^3} \tag{4.26}$$

4.5.4 Application to Graphics

So far we have looked at these models as they exist in their original contexts. The model of Munnannur and Reitz [2007] (and the extensions of Kim et al. [2009]) describes the current state-of-the-art for droplet modelling in this way (although products such as KIVA [Los Alamos National Laboratory 2018] and ANSYS Fluent [ANSYS 2018] favour even older models such as those of O'Rourke and Bracco [1980]). However, the composite nature of Munnannur and Reitz [2007] results in a complex model (incorporating many components from previous approaches) and certain model choices such as the combination of the threshold of Brazier-Smith et al. [1972] with the interaction volume calculation of Ko and Ryou [2005] (that, in turn, is based on the former's competing model of Ashgriz and Poo [1990]) are not necessarily well justified and/or re-engineered for compatibility. Whilst there have been documented applications of this model (such as Kim et al. [2009]) that demonstrate its ability to create meaningful results on a larger scale for spray engineering purposes, we look to develop a more unified model that offers greater control and robustness to inputs than possible with this complex, composite model.

We choose to develop a model based on a subset of the components that we have described here, with a focus on those which will enable our plausible recreation of similar behaviour without some of the model complexity of say Munnannur and Reitz [2007]. We look to introduce these behaviours to enable increased realism to be created over current methods, but as an artistic tool require it be controllable, robust and can integrate easily into our existing systems. These considerations have led to our development of a collection of extensions to these models and are a main theme for discussion for the remainder of the work in this chapter.

4.6 Our Approach

The approach we take is to develop a model to introduce these behaviours into a widely used method for spray simulation in computer graphics (§2.5), the simple ballistic particle system. In this way, we consider a system of droplet particles under the influence of gravity, drag and inter-droplet interaction. By using these discrete droplet collision interaction models we novelly allow each particle in our system to meaningfully represent a single droplet. The model we now describe recreates the coalescence and separating phenomena that occur in reality (§4.4) as well as novel handling of a visible representation of fragmentation and shattering. This approach is the first to use explicit modelling of droplet collisions for computer graphics applications, and to do so considers the combination of the following model components to ensure deterministic, visually consistent and high fidelity results.

This model consists of three key stages: binary collision detection, outcome classification and collision resolution.

- For collision detection, we use sphere-moving sphere intersection tests, approximate neighbourhoods (§4.6.3) and efficient, spatially-organised particle data from OpenVDB Points to deterministically detect collisions between pairs of droplets in reasonable time.

- The outcome of the collision is then determined using the experimentally-validated, physically-based threshold model of Ashgriz and Poo [1990], which we extend to be robust to a full range of input states.

- Finally, to resolve these collision outcomes, we use a combination of momentum/mass conservation, the physically-based fragmentation model of Munnannur and Reitz [2007] and non-uniform initial behaviours for satellite droplets.

We choose to incorporate and extend some of the pre-existing droplet threshold model components as they are physically meaningful, well-cited, and have been demonstrated to show good accordance with real droplet behaviour. We first give an overview of our model below in Algorithm 2. Following that, we will discuss our choice of threshold models from §4.5 and the required extensions for the outcome determination (§4.6.1) and collision resolution (§4.6.2) stages, before describing the approach we take for binary collision detection(§4.6.3).

Algorithm 2 Ballistic Droplet Particle System

Text in blue indicates the additional stages required for our droplet model compared to standard, purely ballistic systems.

```
 1: for all timesteps $t$ do
 2:     force update:
 3:     for all particles $p$ do
 4:         gravity: $\mathbf{u}_p \leftarrow \mathbf{u}_p + \Delta t \mathbf{g}$
 5:         drag: $\mathbf{u}_p \leftarrow \mathbf{u}_p + \Delta t \frac{\mathbf{F}_{drag}}{m_p}$
 6:     end for
 7:     binary collision detection (Algorithm 3)
 8:     for all particles $p$ do
 9:         if collided then
10:             find collision partner
11:             droplet collision resolution (Algorithm 4)
12:         end if
13:         advect: $\mathbf{x}_p \leftarrow \mathbf{x}_p + \mathbf{u}_p \Delta t$
14:     end for
15: end for
```

4.6.1 Robust-to-Input Outcome Classification

We choose to focus our model on the interactions missing in other computer graphics approaches to droplets, and in particular those that uniquely occur for liquid droplets and are the contributing factors to most spray behaviour, i.e. those of coalescence, stretching separation and reflexive separation. As others (in engineering) before us, we ignore the case of droplet bounce as it can depend on atmospheric factors and deformation which we do not explicitly model in our otherwise simple particle system. For the classification of the collision outcome we extend the model of Ashgriz and Poo [1990], offering both stretching and reflexive separation thresholds (Figure 4.6). Whilst this model has been shown to give good results for a variety of inputs as the thresholds are defined in relative terms and are therefore scale independent (as long as the spherical droplet assumption applies), the original work considers only a narrow range of input collision parameters, $0.5 \leq \delta \leq 1$ and $\text{We} \leq 100$. We make some additions to allow the full possible range of inputs that may arise during our simulation.

For high velocities or larger droplets (i.e. large We), we need to be able to handle collisions that surpass both the reflexive and stretching separation thresholds. Similarly, for lower δ and X, we may find collisions where the smaller droplet is fully interacting, which were not considered in the original work (as the experiments used as comparison did not cover these cases). In the case of both thresholds being surpassed (only occurring for lower values of X) we assume that reflexive separation takes precedence over stretching separation as this behaviour better describes head-on collision. For fully interacting smaller droplets, we cap the original formulation for ϕ_j to assume full interaction (i.e. $\phi_j = 1$) and calculate the interacting region ϕ_i using the geometric formulae for spherical segments and caps. Without this modification to the interaction volume calculation, the formulae given by the original paper (Equations 4.10 - 4.11) give non-physical results (Figure 4.7) when the smaller droplet is overlapped by the larger one.

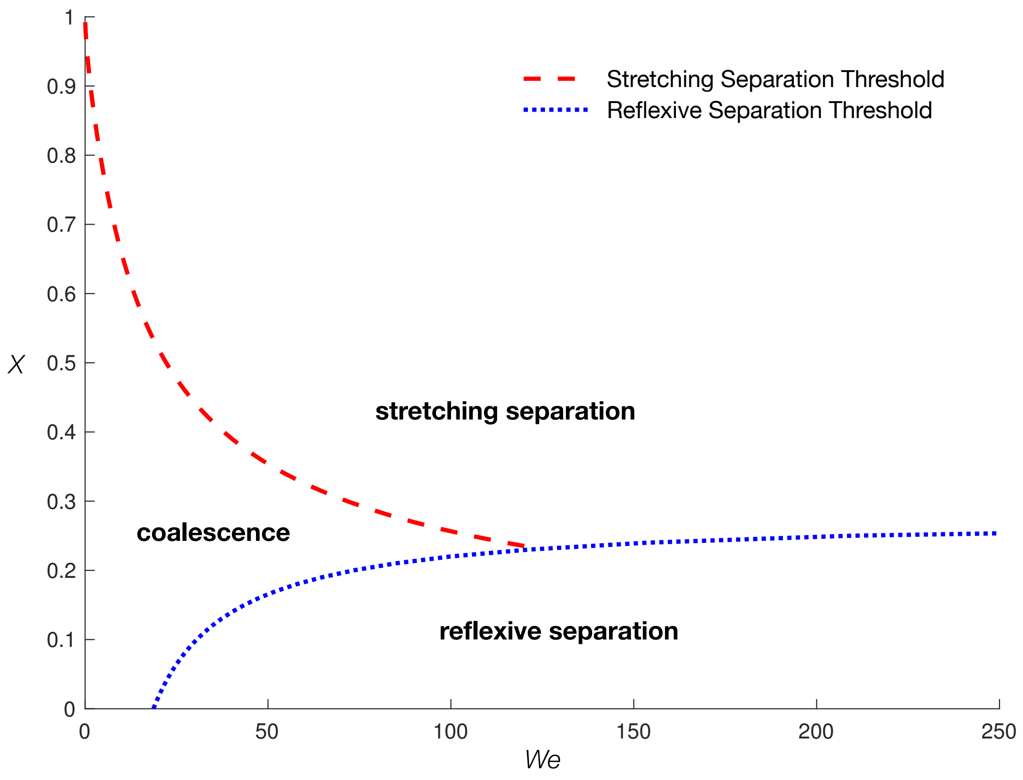


Figure 4.6: Our collision thresholds (for $\delta = 1$) based on those of Ashgriz and Poo [1990], with reflexive separation taking precedence for low impact number collisions.

4.6.2 Visually Plausible Post-Collision Characteristics

To consider the visible outcome of these collisions requires methods to resolve these collisions in plausible and temporally consistent ways. An important component of this is the time at which to resolve the collision, the modifications made to the system should not introduce any artefacts such as popping or flickering. We describe how to calculate this time using the same approach as for detection of these collisions in the next section.

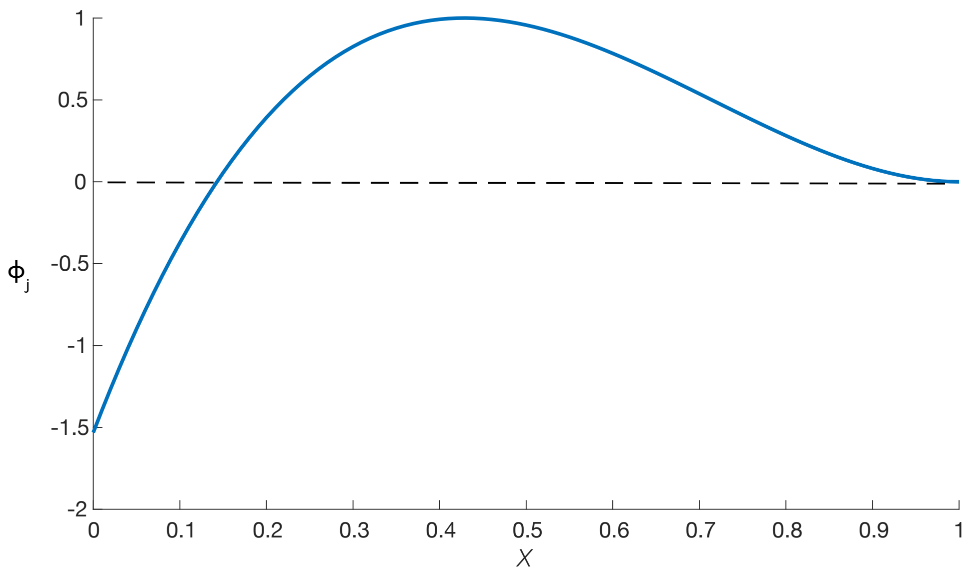


Figure 4.7: ϕ_j (Equation 4.11) against X for $\delta = 0.4$, note that as $X \to 0$, the volume fraction ϕ_j in fact reduces as the entire droplet should be overlapped i.e. $\phi_j = 1$.

For coalescence collisions, mass and momentum conservation (Equations 4.14 - 4.17) are used to combine the colliding droplets at the time of resolution.

For separation collision scenarios, elements from the works of O'Rourke and Bracco [1980], Tennison et al. [1998] and Munnannur and Reitz [2007] are combined to calculate the velocity updates for the incident droplets and if required, the number of satellites to be created. Whilst the method of Munnannur and Reitz [2007] covers the number of satellites created, their modelling of the collisions in isolation rather than in a larger system like ours, requires no consideration of satellite creation behaviours or post-collision characteristics. Even works considering spray simulation using these models, such as Kim et al. [2009], separate satellites from the rest of the system and so are able to consider simplifications such as uniform creation criteria. To keep our particle system closed whilst minimising visual artefacts we require non-uniform satellite creation criteria in fragmenting interactions.

Satellite creation

Using the assumption that the satellites created are of uniform radii (due to the cylindrical nature of the ligament), we emit the satellites evenly along this nominal ligament with velocities that replicate the stretching behaviour causing its break-up.

We then calculate velocities using momentum conservation with a weighted bias to their closest parent droplet, such that for satellite $n = 1, ...n_{sat}$, we have:

$$\mathbf{u}_n = \frac{V_{lig,i}}{V_{lig}}\tilde{\mathbf{u}}_i + \frac{V_{lig,j}}{V_{lig}}(\tilde{\mathbf{u}}_j + (1 - \frac{2n}{n_{sat}+1})\tilde{\mathbf{u}}_{ij}) \tag{4.27}$$

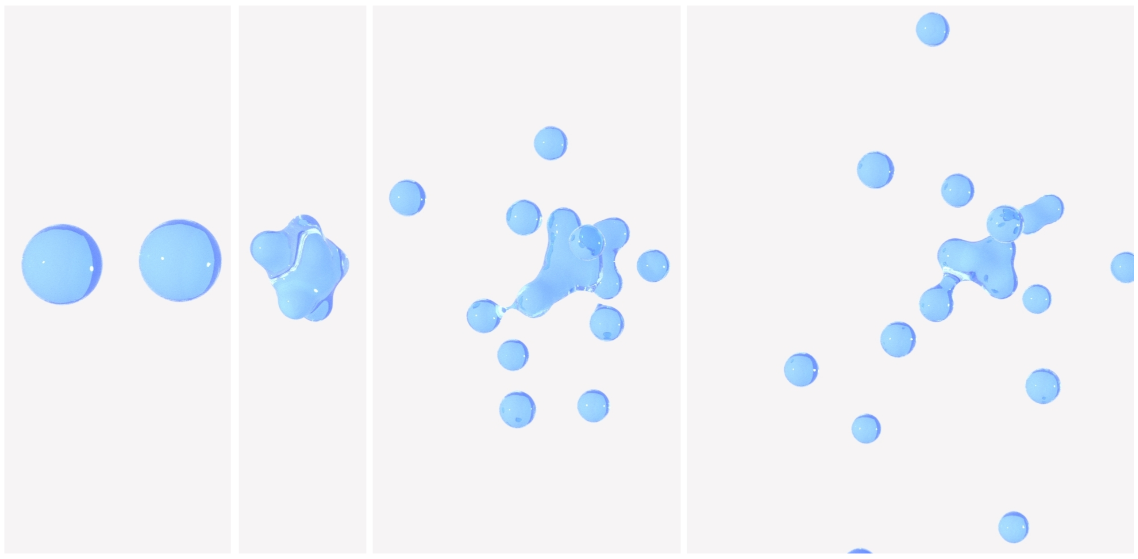


Figure 4.8: Shattering reflexive separation (left to right), greatly increased kinetic energy version of reflexive separation seen in Figure 4.3, creating many satellites with perturbed satellite velocities (§4.6.2).

with $\tilde{\mathbf{u}}_k$ for $k = i, j$ the post-collision velocity of the parents (Equation 4.18), $V_{lig,k}$ for $k = i, j$ the contribution of droplet k to the ligament volume and satellite n_{sat} is that closest to droplet j. This is able to mimic the stretching and separation phenomena that occur in reality (Figure 4.3).

Perturbation and shattering

When droplets collide at high energy, the interacting droplet mass undergoes large deformations and oscillations, perturbing the velocities of the satellites that are created as a result. As the fragmentation model from Munnannur and Reitz [2007] provides no information on torque, to approximate this behaviour, we add a random rotation to the velocity of the satellites. We apply a rotation R_n to each satellite (including updated colliding droplets in reflexive separations) such that their final velocity is given by $\tilde{\mathbf{u}}_n = R_n \mathbf{u}_n$. The magnitude of this rotation is a uniformly sampled value in $\alpha = [0, \eta n_{sat}]$ where η defines a user parameter to define the scale of the rotational variation introduced by these deformations, and the perturbation increases with the number of satellites being created. We further scale R_n by $(1-\alpha)^2$ such that larger perturbations also dissipate more energy. In this way, for extreme cases, we are also able to approximate the behaviour of reflexive separating collisions as they tend to shattering [Pan et al. 2009] (Figure 4.8).

4.6.3 Binary Collision Detection

Given our models for the prediction and resolution of the outcomes of a droplet collision we must first find these collisions as they occur in our system. Early applications of these models such as O'Rourke [1981] track parcels - collections of droplets, rather than individual droplets themselves and use stochastic methods to determine both the presence of collisions and their parameters such as X. As we are considering visible phenomena of droplet interaction such an approach could be troublesome. Given our spherical droplets we are able to instead use a collection of standard collision detection methods to find collisions that occur between each droplet individually. In this section we will describe these collision detection schemes between spheres.

Discrete overlap detection

The simplest method of collision detection is to test for a collision between droplets at a single time t. That is, testing whether the radii of the colliding droplets overlap. For this simply compare the sum of the radii to the distance between the droplets i.e. for droplets i and j with radii r_i and r_j and position $\mathbf{x}_i(t)$ and $\mathbf{x}_j(t)$ resp. Then, the spherical droplets are overlapping i.e. colliding, if:

$$||\mathbf{x}_i(t) - \mathbf{x}_j(t)|| < r_i + r_j \tag{4.28}$$

Whilst this method is extremely simple and can be relatively efficient (using comparisons of the squares of both sides to avoid costly square root operations), no further information is given about the collision, simply that they are colliding at time t. If we require more details on the collision, we can look to using a continuous collision detection scheme instead.

Continuous collision detection

Rather than considering the position of the particles at a single time t, consider the trajectories of particles over the arbitrary interval $[t, t + \Delta t]$. Suppose the particles move through this interval linearly from their initial positions, $\mathbf{x}_i(t), \mathbf{x}_j(t)$, along their velocities $\mathbf{u}_i(t), \mathbf{u}_j(t)$ at time t. Now suppose there exists a hypothetical time where these two spheres are just overlapping (equality in the above equation), say $t + \tilde{t}$. At this time, the positions of the particles are given by $\mathbf{x}_k(t + \tilde{t}) = \mathbf{x}_k(t) + \mathbf{u}_k(t)\tilde{t}$ for $k = i, j$ and they are separated by the sum of the radii of the particles s.t. we have:

$$||(\mathbf{x}_i + \mathbf{u}_i\tilde{t}) - (\mathbf{x}_j + \mathbf{u}_j\tilde{t})|| = r_i + r_j \tag{4.29}$$

Taking the square of both sides of the above equation we arrive at the quadratic equation for $\tilde{t}$:

$$\mathbf{x}_{ij}^2 + 2\mathbf{u}_{ij}\mathbf{x}_{ij}\tilde{t} + \mathbf{u}_{ij}^2\tilde{t}^2 = (r_i + r_j)^2 \tag{4.30}$$

(where vector-vector multiplication is the dot product), this then can be rearranged into

$$\mathbf{u}_{ij}\tilde{t}^2 + 2\mathbf{u}_{ij}\mathbf{x}_{ij}\tilde{t} + \mathbf{x}_{ij}^2 - (r_i + r_j)^2 = 0 \tag{4.31}$$

From this equation, the (real) solutions describe the times at which the spheres moving along the lines through $\mathbf{x}_i(t), \mathbf{x}_j(t)$ with linear trajectories defined by $\mathbf{u}_i(t), \mathbf{u}_j(t)$ are just in contact (separated by the sum of their radii), that is the initial collision time (smaller root) and the separation time (greater root). If these roots do not exist, or are both larger than the interval end $t + \Delta t$, then the particles are not found to collide.

Similar to problems resolving contacts in rigid bodies [Vouga et al. 2017], by trying to resolve all collisions in pairs we could end up requiring large iteration numbers and very small substepping as each resolved collision alters the system, possibly causing subsequent collisions etc. As such, because we are not explicitly enforcing incompressibility per particle in the usual sense but instead focusing on resolving particular behaviours, we choose to limit each particle to at most one resolvable collision per timestep. To ensure as much temporal consistency as possible, we make sure to choose the initial collision that should occur for each particle. That is, for each particle i, we test all of its neighbours in $N_{trajectory,i}$ and store a reference to the corresponding particle with which it has the lowest initial collision time, i.e. lower root t^{-}. We do this for each droplet in the system and follow with a subsequent pairing step, as in the particle merging scheme of Ando et al. [2012], to check each particle and its preferred collision pair. If both particles refer to the other and their collision time is within the substep Δt, a binary collision is successfully found and the pair are marked and removed from the next iteration of detection.

Algorithm 3 Binary Collision Detection

```
1: for all iterations do
2:     detection:
3:     for all particles $p$ do
4:         $t_{p,coll} = \Delta t + \epsilon$
5:         for all particles $q$ in $N_p$ do
6:             calculate $t^{-}$ (Equation 4.31)
7:             if $t^{-} < t_{p,coll}$ then
8:                 set $collId_p = q$
9:             end if
10:        end for
11:    end for
12:    pairing:
13:    for all particles $p$ do
14:        if $collId_p = q$ and $collId_q = p$ then
15:            keep collision
16:        else
17:            return $p$ and $q$ to candidate points
18:        end if
19:    end for
20: end for
```

Neighbourhood restriction

Given the above detection methods, in practice a brute force implementation that considers collisions between all particles in the system would be extremely costly ($O(N^2)$), especially for larger particle counts. As such, we can avoid this by restricting these collision tests to particles within local neighbourhoods of each other.

We consider the neighbourhoods around each particle and only test against particles within this neighbourhood to reduce redundant calculation. Consider a neighbourhood around a particle i. Suppose we know the radius, r_i, but do not know the radii of the surrounding particles r_j. Then for the discrete collision detection scheme above we define the neighbourhood around particle i as

$$N_{overlap,i} = \{\text{particles } j \text{ s.t. } ||\mathbf{x}_i - \mathbf{x}_j|| < r_i + r_{max}\} \tag{4.32}$$

where r_{max} is the maximum radii of any droplets in the system.

Similarly for the continuous collision detection scheme, we define

$$N_{trajectory,i} = \{\text{particles } j \text{ s.t. } ||\mathbf{x}_i - \mathbf{x}_j|| < r_i + r_{max} + \Delta t(||\mathbf{u}_i|| + u_{max})\} \tag{4.33}$$

where u_{max} is the maximum speed of any droplets in the system.

Then we can also use two tests from Nordin [2001] to reduce this further. These tests require that the two particles are travelling towards each other:

$$||\mathbf{u}_{ij}(t)|| < 0 \tag{4.34}$$

and that the relative displacement is greater than the distance separating the two particles:

$$||\mathbf{u}_{ij}(t)||\Delta t > ||\mathbf{x}_{ij}(t)|| - (r_i + r_j) \tag{4.35}$$

Discrete detection with continuous ranking for binary collisions

The above continuous collision detection method is able to find all possible collisions that can occur, even considering high velocity, large timesteps or very small particles. However, the size of the neighbourhood $N_{trajectory,i}$ can make these calculations very costly. We have found that it is sufficient to use an approximation of this neighbourhood around the substep interval, $N_{overlap,i}$ given by

$$N_{overlap,i} = \{\text{particles } j \text{ s.t. } ||\mathbf{x}_i - \mathbf{x}_j|| < r_i + r_{max}\} \tag{4.36}$$

All of the methods discussed here can be combined when considering fast moving, sparse particle sets to try and reduce excess computation. For example we could first consider using a number of iterations of detection within $N_{overlap,i}$ to reduce the number of candidate particles before testing $N_{trajectory,i}$. However, in practice we found that even at low iteration counts (< 5) we are able to find a significant number of collision pairs using only the narrower neighbourhood, for most reasonable timestep sizes.

4.7 Implementation

This model has been implemented into the Dynamo simulation framework at DNEG, built upon OpenVDB Points as a plugin for Houdini. Each stage of the algorithm is implemented as a multi-threaded operation, parallelised over the leaves of the OpenVDB Points grid.

Algorithm 4 Droplet Collision Resolution

```
1: for all colliding droplet pairs $i$, $j$, s.t. $r_i \geq r_j$ do
2:     if We > We$_{reflex}$ then
3:         resolve reflexive separation:
4:         Update $\mathbf{u}_i, \mathbf{u}_j$ (Equation 4.18)
5:         Solve for $r_{sat}$ (Equation 4.25)
6:         Calculate $n_{sat}$ (Equation 4.26)
7:         if $n_{sat} > 2$ then
8:             Emit $n_{sat}$ satellites (Equation 4.27)
9:             Update $r_i, r_j \leftarrow r_{sat}$
10:        end if
11:    else if We > We$_{stretch}$ then
12:        resolve stretching separation:
13:        Update $\mathbf{u}_i, \mathbf{u}_j$ (Equation 4.18)
14:        Calculate $V_{lig}$ (Equation 4.21)
15:        Solve for $r_{sat}$ (Equation 4.25)
16:        Calculate $n_{sat}$ (Equation 4.26)
17:        if $n_{sat} > 0$ then
18:            Emit $n_{sat}$ satellites (Equation 4.27)
19:            Update $r_i, r_j$ using mass conservation
20:        end if
21:    else
22:        resolve coalescence:
23:        merge droplets $i$ and $j$, update $\mathbf{u}_j$ and $\mathbf{x}_j$ (Equations 4.14 - 4.17)
24:    end if
25: end for
```

Table 4.1: Suggested values for the parameters considering material properties of water/droplets. The relative nature of the collision parameters makes these somewhat flexible and allows artistic control through their modification e.g. modelling the approximate behaviour of larger 'droplets' than those in reality (Figure 4.12)

Name	Description	Default
σ	surface tension coefficient (N/m)	0.072
ρ	liquid density (kg/m^3)	997.044
α	drag coefficient	0.0001
Δt	timestep size (s)	1/24
r_{max}	maximum droplet radius (mm)	$10-100$
r_{min}	minimum droplet radius (mm)	$0.05-0.5$
η	satellite velocity perturbation	0.01
t_{delay}	minimum time between collisions (s)	1/24
n_{max}	maximum satellites created per collision	5
r_0	initial droplet radius (mm)	1

- *Solving Equation (4.25):* We use `bracket_and_solve_root` in the Boost library [Schäling 2011] with an initial guess of the solution to the 3rd order polynomial

$$\beta \mathrm{We}_0^{\frac{1}{2}} \left(\frac{r_{bu}}{r_0}\right)^3 + \left(\frac{r_{bu}}{r_0}\right)^2 - 1 = 0 \tag{4.37}$$

- *Neighbourhood search:* The inherent spatial organisation of the point data structure allows a fast and easy implementation of the collision detection system that we describe in this paper, without the requirement for any other acceleration structures.

- *Time of resolution:* Collisions are resolved at the impact time, i.e. $t_X = t^- + \frac{t^+ - t^-}{2}$, when the droplets are most overlapping. We cap this to some max value, i.e. Δt, for collisions with very large t_X. Droplets are advanced to this time, resolved and then back-advected with their post-collision velocities through t_X, such that all droplets are in the correct positions accounting for their mid-timestep velocity change after they are advected as normal through Δt.

- *Rest between collisions:* Each droplet, once collided is deactivated from collisions for a few timesteps. This allows ligament breakup to fully resolve and satellites to separate, avoiding over detection of collisions and excessive merging in slower moving scenes.

- *Droplet sizes:* To avoid over-merging of droplets and/or the introduction of very small droplets we clamp the droplet sizes to a user defined range. We skip any coalescence/fragmentation that would produce droplets outside this range but continue to allow large droplets to fragment, small droplets to coalesce etc.

- *Drag:* To best represent the effects of the varying size distribution of droplets on their motion, we use a simple size-dependent drag force. For application to our simple particle system we make assumptions and modifications to the usual drag formulation. Here, we assume that magnitude of the background air velocity is negligible compared to the droplet velocity and replace the usual $\Delta \mathbf{u}$ term with $-\mathbf{u}$. Then for simplicity, combine all other terms (including terms from the mass, m) into a single user parameter $\alpha > 0$ (as in Song et al. [2005]) and the choice of exponent σ such that the velocity update is as follows:

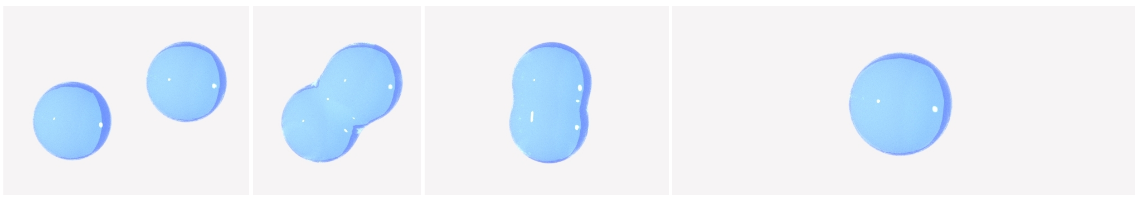
$$\frac{d\mathbf{u}}{dt} = -\frac{\alpha}{r^\sigma} \mathbf{u}^{3-\sigma} \tag{4.38}$$

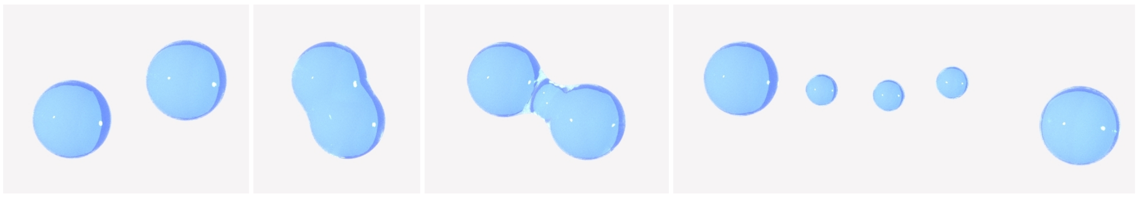
  where $\sigma \in 1, 2$ such that the drag equation mimics either the Newton or Stokes drag for larger and smaller scale simulations respectively. As in Mihalef et al. [2009], these could also be combined with an interpolation scheme but we have not found the need to do so.

- *Particle surfacing:* For close-up examples, to better demonstrate ligament stretching and breakup using our particles we rasterise particles into a VDB level-set, apply smoothing operations (as described by Museth [2014]) and subsequently convert to a mesh for rendering, all using the existing VDB operators in Houdini (more on this in Chapter 5).

Table 4.2: Simulation times for results shown. The complexity of the operations, including the emission and deletion of particles, and widely varying particle count arising from interactions means results can differ considerably. All simulations performed with two 8-core 3.10Ghz Intel Xeon CPUs and 64GB RAM.

Example	Frame time w/o collision	with collision	Final # particles w/o collision	with collision
Figure 4.11 (our model)	0.025s	0.028s	200	139
Figure 4.11 (FLIP)	0.055s	N/A	200	N/A
Figure 4.11 (SPH)	0.006s	N/A	200	N/A
Figure 4.11 (RBD)	0.006s	N/A	200	N/A
Figure 4.12 (second from top)	0.101s	0.277s	111,532	181,766
Figure 4.12 (second from bottom)	0.101s	0.136s	111,532	57,846
Figure 4.12 (bottom)	0.101s	0.099s	111,532	22,265
Figure 4.13	1.35s	1.89s	3,235,474	1,332,316


(a) Coalescence: We = 16.


(b) Stretching Separation: We = 447.

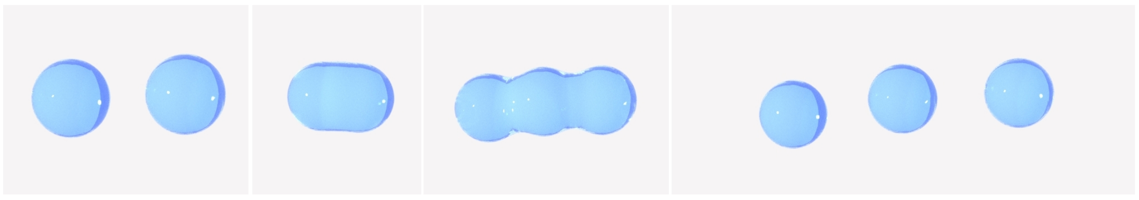

(c) Reflexive Separation: We = 155.

Figure 4.9: Simulated results of different droplet collision outcomes, using our method.

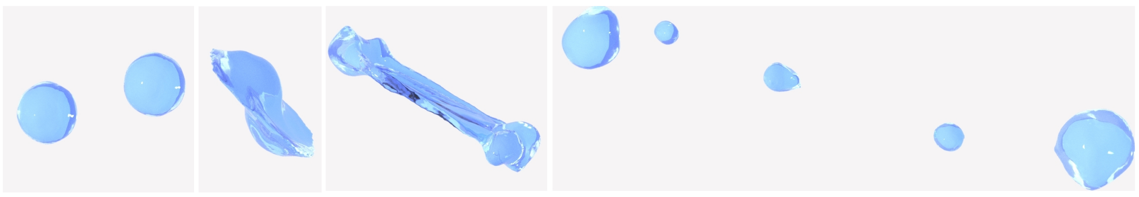


Figure 4.10: Stretching separation between droplets displayed by a FLIP simulation. Requires 18k particles to replicate behaviour that we simulate with a single pair of particles (Figure 4.9b).

4.8 Results

The results in this section describe the stand-alone application of our droplet model into a simple particle system. By introducing these interactions into these systems, we are able to plausibly and efficiently model large systems of water droplets. We first demonstrate the simplest, fundamental case of a single binary droplet collision in Figure 4.9, showing how our model approximates ligament formation and handles the topology changes that occur during the interactions described in §4.4. We compare our stretching separation result to a recreation using Houdini's FLIP liquid solver in Figure 4.10, which we found requires a system of at least 18k particles to demonstrate the same stretching/fragmenting phenomena (albeit with added qualities eg. surface deformation).

In Figure 4.11, we show a close-up of a small system of droplets to further demonstrate the interactions that occur. Comparisons are given to the attempted use of FLIP, SPH and rigid body solvers (native to Houdini) to create similar interactions for the same input particle set. The coalescence and fragmentation that we model create levels of detail and realistic liquid features (varying size droplets, stretching of ligaments) that are not created by other methods, demonstrating their lack of a meaningful one-to-one mapping between droplet and particle. This result also demonstrates the potential for this method to be used for fast simulation of small-scale droplet interactions but further additions would be required to overcome the spherical uniformity of the particles and merging behaviours at this scale, such as for bubbles in Patkar et al. [2013].

Following this, we demonstrate the effect of our model on larger-scale spray effects that are often simulated with ballistic particle systems. To best illuminate the specific effects of our model we restrict these examples to an otherwise purely ballistic system. However, in practice, our model can easily be combined with other available tools used with these systems such as noise fields, droplet age and other external forces. In Figure 4.12 we simulate a geyser-like vertical jet of liquid droplets. In this example, the introduction of these mechanisms for droplet merging and splitting increases visual quality and reduces artefacts seen in the ballistic particle equivalent, here from a poorly constructed emitter creating uniformity that persists due to lack of any interaction. The other panels of this figure further demonstrate the influence of parameter variation on our model, controlling the tendency to coalesce/fragment and varying the resulting droplet distribution between an finer spray and fewer, larger droplets.

Finally, we use this model to approximate very fine-scale spray effects, considering millions of particles in the fountain example of Figure 4.13. Here, uniform radius particles are initially perturbed by a small amount of curl-noise [Bridson et al. 2007] to vary the flow and allow interactions. Whilst largely dominated by coalescence, the other interactions in this example allow the system to maintain a range of droplet sizes which, in combination with the size-dependent drag force, break-up the uniformity of the flow and create a realistic fountain effect.

These larger simulation examples demonstrate that even in scenarios where droplet size and individual interactions are not as easy to directly recognise, their effect on the droplet distribution continues to improve the quality of the results, adding an increased sense of realistic motion and scale.

4.8.1 Limitations

Modelling interactions between particles increases simulation time over simple ballistic particle tracking, in particular due to the collision detection calculations. In practice, we find that the approximate neighbourhood method, the use of spatially organised particle data, and often reduced particle count brings this cost down to a manageable level for the increase in quality that is gained (Table 4.2), but other improvements could be made. It should be noted the implementation used to generate the results in this section demonstrates a first attempt at using OpenVDB Points for neighbourhood calculations and so there are likely further optimisations to be had.

The discrete threshold model and fragmentation calculation are relatively complex with a lot of different components (albeit more so in model understanding than usability or computational complexity). We choose these methods as they are physically meaningful and shown to be accurate to real observations, but there may be simplifications possible in the future. Furthermore, whilst there are a number of parameters that arise out of the model, liquid density, surface tension, shattering perturbation and the limits on size and number of satellites, they have quite intuitive relationships with the results and physical parameters like surface tension offer control over the look/relative scale of the results.

The scale of interactions we capture allows high fidelity, detailed results for sprays composed of visible droplets to be created and whilst the overall topology is also captured at a microscopic scale (Figure 4.11 - top), the deformations of the droplet surfaces are not. This could impede the model's practicality for close-up results. To overcome this, it may be interesting to consider the deformation and the oscillating dynamics of the droplet surfaces using an approach like spherical harmonics [Ashgriz 2011].

Finally, as this model defines droplet-like interactions specifically, in areas where the droplet approximation breaks down, i.e. very dense sprays or bodies of liquid, we are unable to capture larger-scale gas-like or liquid-like motions. However this could be overcome by appropriate coupling of this method to larger-scale fluid motion such as a free-surface solver like FLIP/PIC.

4.9 Summary

In this chapter, we have introduced a model for physically-based interaction to be incorporated into particle systems of liquid droplets and sprays. Our model is the first of its kind to consider the visible impact of the various separating droplet phenomena that occur in reality, and in doing so presents a novel physically-based method to define a complete system of merging and splitting phenomena between droplets. The results that we provide demonstrate an increased sense of scale and physicality from the inclusion of these interactions, creating varied and evolving droplet size distributions and increased small-scale details from fragmentation. Our model recreates plausible behaviours of droplets and sprays on both small and large scales, capturing localised interaction in scenarios where existing fluid techniques fail and helping the creation of larger-scale effects when combined with our size-dependent drag force. Whilst this has yet to be integrated into production, upon showing this model and its results to artists at DNEG they responded with:

"Having ballistic particles is not enough for a believable whitewater solution and I have no problem in seeing the potential in having this available" - Ole Eidsheim, FX Artist.

"I see more and more fluid sim methods becoming more accurate, as the CPU/GPU limitations are removed. I am excited how this droplet method could be implemented in a future production workflow." - Andy Guest, FX Artist.

By instilling missing physical characteristics into our system, we have allowed more realistic phenomena to be recreated and we expect that similar physically-motivated approaches would also be useful for other secondary fluid phenomena. We forsee the logical extension of this work to be the coupling of our small-scale interactions with larger-scale fluid behaviours in the creation of a more comprehensive splash/spray system. Other work could also look to build on this model to incorporate missing physical behaviours (i.e. surface deformations, continuous merging) to allow fast simulation of close-up droplet effects.

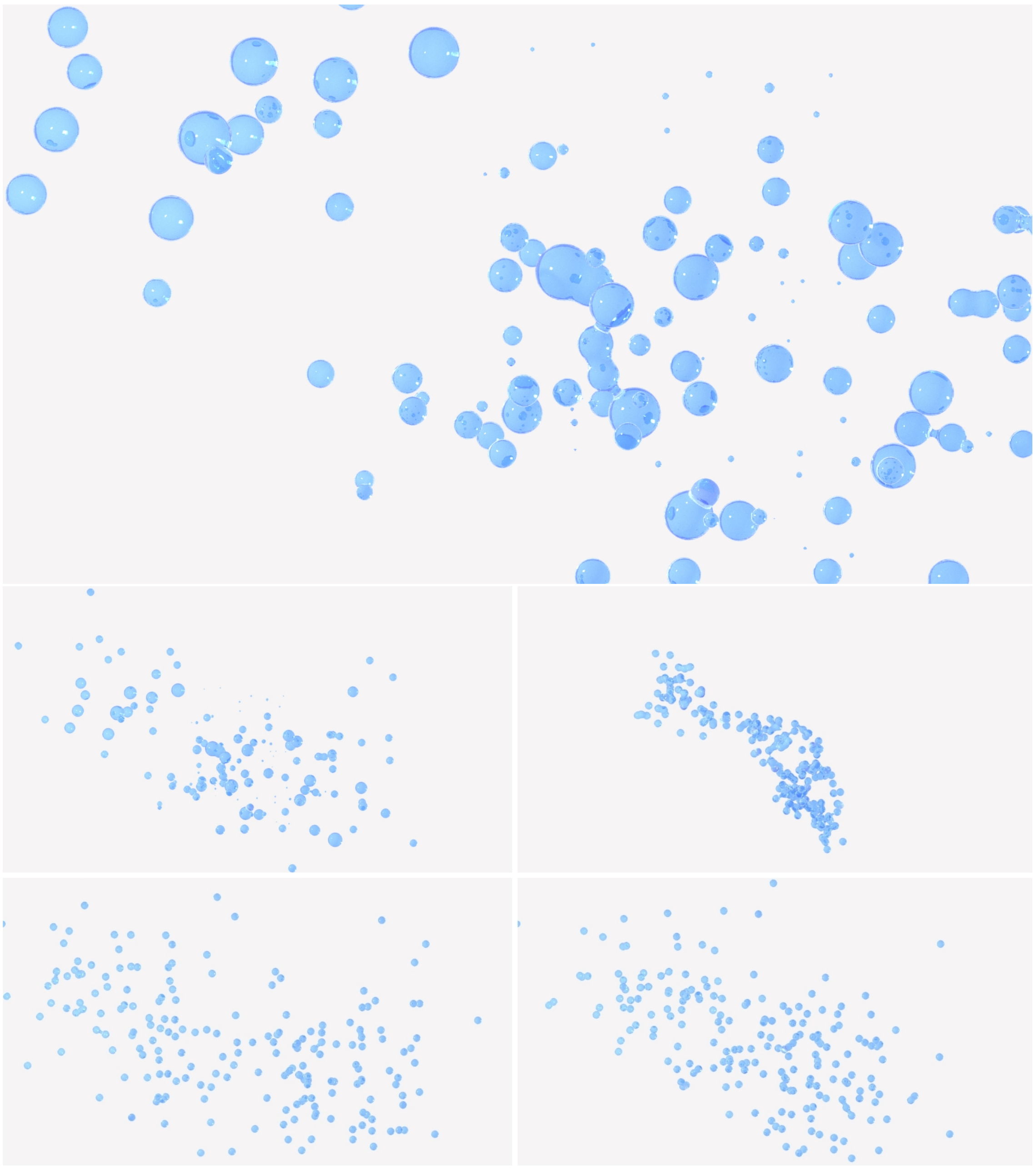


Figure 4.11: Small-scale system of droplet interactions: close-up using our droplet collision model (top); zoomed out view using our model (middle left), using FLIP solver (middle right), SPH solver (bottom left) and RBD solver (bottom right). While FLIP does create larger-scale motions, other systems are unable to create realistic small-scale droplet characteristics, e.g. size variation, ligament formation and fragmentation, at low particle density.

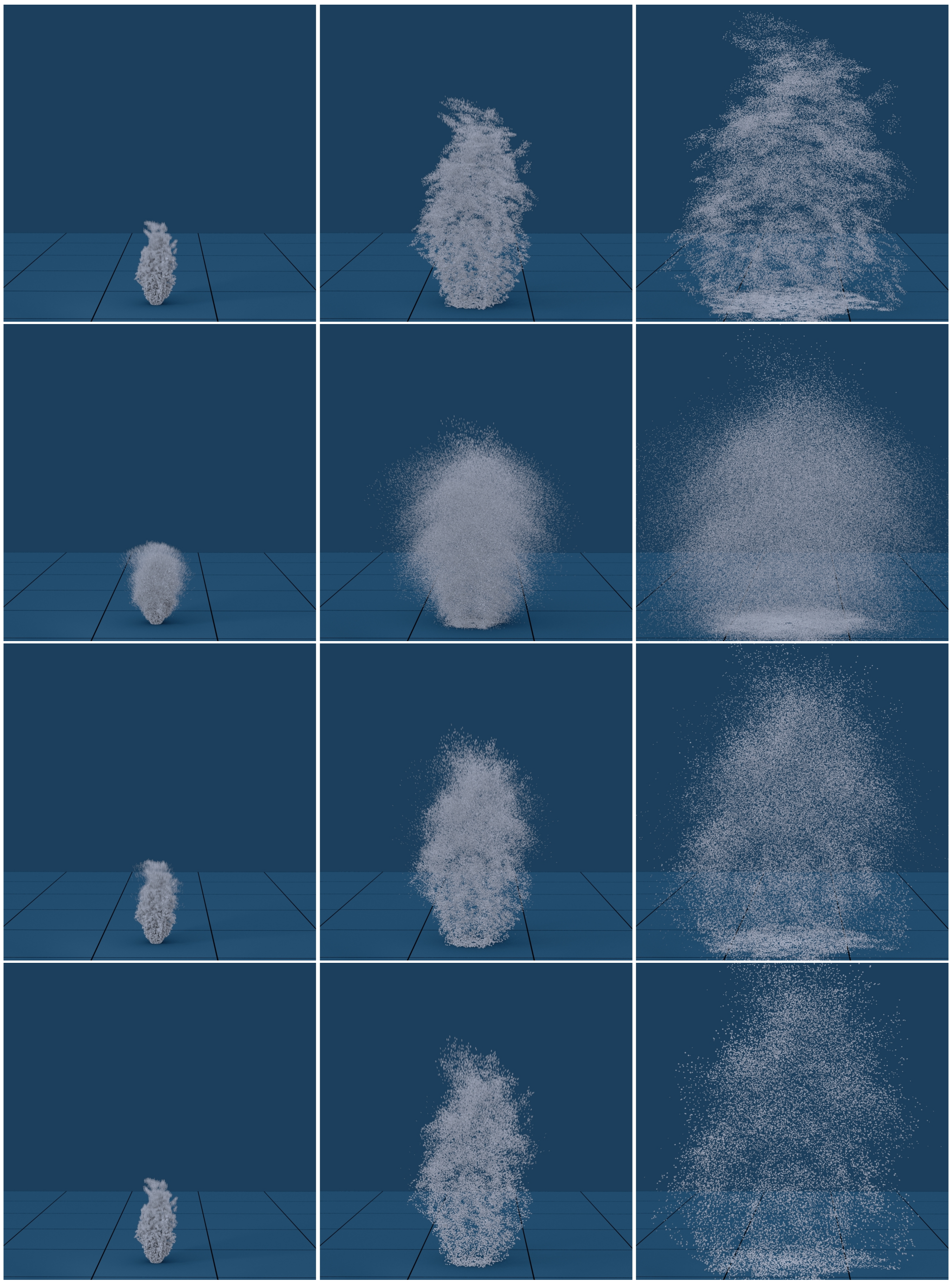


Figure 4.12: Geyser simulation (left to right): purely ballistic particle system - top, our model with reduced surface tension (×0.1) - second from top, our model - second from bottom, our model with increased surface tension (×10) - bottom. Low surface tension interactions cause many satellites to be emitted and create a finer spray effect, whilst high surface tension tends to coalescence and causes system to tend to fewer, larger droplets.

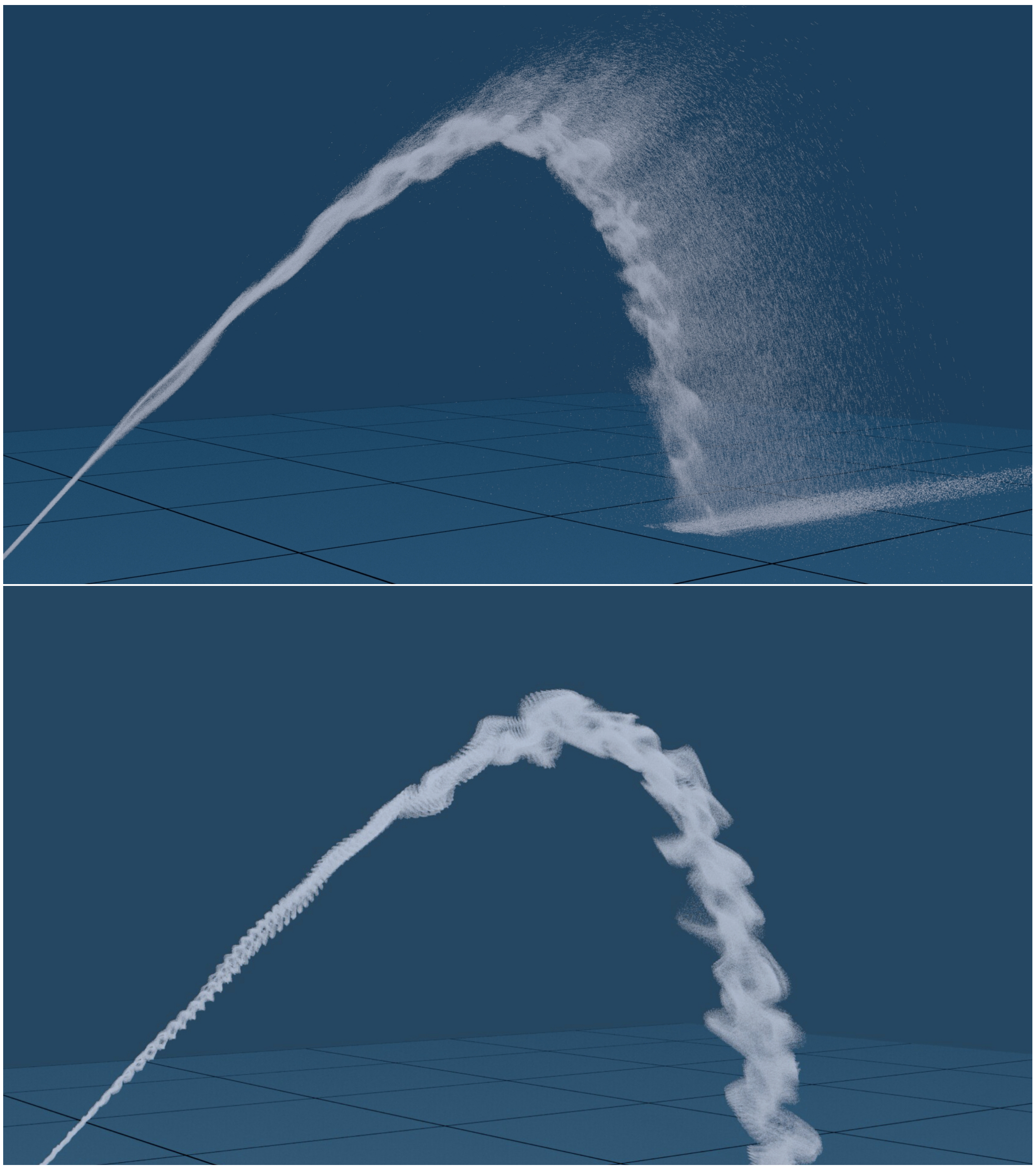


Figure 4.13: Fountain-like jet (1.3m particles): Top - Droplet interactions create a varied droplet size distribution and therefore drag strength, which causes larger-scale break-up of the flow. Bottom - No interaction exposes artefacts in emitter and timestepping, these would usually require artist intervention and re-simulation to fix.

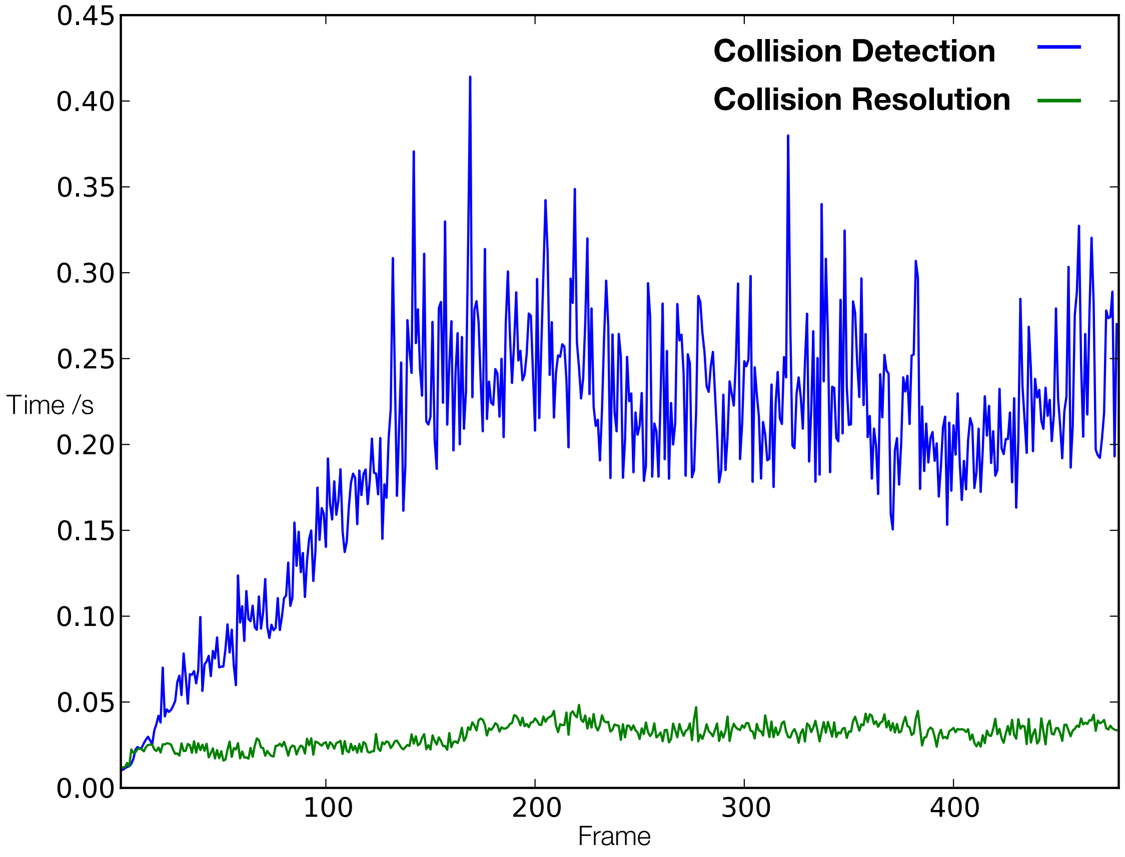


Figure 4.14: Times of collision detection and collision resolution (incl. outcome determination and satellite emission) operations for simulation in Figure 4.13. 480 frames with 2 substeps per frame, $\Delta t = 1/96$.

Chapter 5

Preserving Detail: Surfacing of Splashes with Droplets

The tool developed in this chapter has become a key component in the liquid (and general particle) effects toolkit at DNEG, being used by artists on at least 15 different films in production including Venom [Fleischer 2018], The Kid Who Would Be King [Cornish 2019], Godzilla: King of the Monsters [Dougherty 2019]. The original implementation of the method of Yu and Turk [2013] was the work of other developers at DNEG, but the extensions to handling uneven particle distributions are my own.

5.1 Overview

Following the task of creating a particle-based simulation of a liquid, there remains the issue of creating a renderable result out of these particles. As with many of the other processes in the liquid effects pipeline, whilst a variety of methods exist, in practice, all require a large amount of artistic input to create a high-quality result. In this chapter we will describe the development of a production-tested, high-quality particle surfacing method; capturing smooth surfaces, splashes and droplets in a single process and so reducing requirements on post-processing to create final-quality results. The contribution of this work is the extension of a state-of-the-art method for increased robustness to the varied nature of inputs and uses that occur in production. This is achieved by introducing a new method to automatically and adaptively calculate values for an otherwise unintuitive user scaling parameter using geometric properties of matrix transformation for volume preservation. This parameter is found to be extremely important to achieving high quality results and our extension makes this method easier to use in the general case of data from different simulation methods.

5.2 Motivation

In the previous sections we have described various methods to create simulated liquid phenomena using particles. By using highly efficient frameworks such as Dynamo and OpenVDB, the creation of massive particle sets ($O(1e^9)$) describing liquid motion has become feasible for production use. Furthermore particle-based simulation frameworks have been shown to be applicable to a large range of liquid phenomena including droplets and sprays (Chapter 4). However, once these simulations have been run, the non-trivial task of creating renderable liquid effects from these particle sets remains.

Standard practice for particle-based simulations of free-surface liquids (both FLIP/PIC and SPH) is to convert the final particles to either a SDF or a mesh to be rendered. Following this, splash and spray simulations (either using methods such as in Chapter 4 or simple ballistic particle systems) are often rendered in separate passes, with droplets being rendered as simple spheres or disks and further elements like fine mists being rendered using volumetric techniques. Finally these components will then be composited together to make a final liquid effect (§2.5).

The initial act of creating a high-quality liquid surface from a particle simulation is a non-trivial one. To capture fine details (in splashes and more turbulent areas) whilst also creating smooth surfaces in less dynamic areas is particularly challenging. Standard practice in production is to use a collection of tools with a user-driven approach in the style of Museth [2014]. In this way, an initial surface will be created using either simple sphere stamping or some more complex method such as the position averaging method of Zhu and Bridson [2005], this will then be followed by the application of a collection of other post-processing steps to create the final renderable result. Combining SDF operations such as smoothing, dilation and erosion in this way (and even sometimes further mesh-based operations such as texture displacement) gives a huge amount of artistic control over the result. However, whilst controllable, such an approach requires a large amount of artist input and tweaking, e.g. requiring layering of these post-processes with user-defined masks etc. to avoid over smoothing of interesting areas.

This is particularly troublesome when trying to retain detail in splashes from high resolution simulations. Whilst able to capture more exciting and plausible behaviour, higher resolution simulations require higher resolution surfaces to maintain the increase in quality. These can be very unwieldy and computationally demanding to work with, even using relatively simple level-set operations. Furthermore, post-processing workflows often lose fine details such as droplets and thin sheets, even at high resolutions (possibly motivating further secondary simulations to recover lost detail). Such demands suggest that any reduction in the amount of post-processing required would be hugely beneficial.

5.3 Related Work

There exist many approaches to creating renderable surfaces from liquid simulations, including mesh tracking [Wojtan et al. 2009; Chentanez et al. 2015b] and implicit surface tracking [Enright et al. 2002, 2005]. We focus on the process of creating a high quality liquid surface *after* the initial particle simulation.

Particle to surface: The most popular approaches are those that create implicit surfaces from these particles. Various methods have been used to do this. Blinn [1982] describe the first method for creating smooth, connected surfaces from point sets using the iso-surface of a scalar field by combining per-point radial kernel functions. By evaluating an iso-surface i.e. $\phi = c$

for some constant c where $\phi(\mathbf{x}) = \sum_p f(\mathbf{x}_p - \mathbf{x})$ is the scalar field at $\mathbf{x}$ for some continuous, monotonically decreasing function $f()$, we arrive at a surface that wraps the particles and creates smooth joining regions between them. This can create liquid-like smooth surfaces from particle sets and is popular for simulations such as SPH [Müller et al. 2003]. However in general, this method suffers from kernel dependence and can create thickening as particles become too close together. Therefore unlike SPH simulations that enforce constraints on the particle distribution, for particle sets coming out of FLIP/PIC simulations (or other arbitrary particle simulations) this can create artefacts on the resulting surface.

A similar approach, that sacrifices the smoothness between particles but avoids issues with thickening or bulging is given by a simple spherical distance function i.e. $\phi(\mathbf{x}) = \min_p(||\mathbf{x}_p - \mathbf{x}|| - r_p)$ where r_p is the radius of the sphere for particle p. This is considered as the starting point for the artist-driven, post-processing approach described by Museth [2014]. This details a workflow based around quickly and easily creating an initial surface and relying on post-processing (i.e smoothing, eroding, dilating) to create a result of renderable quality. In production, this post-processing workflow has become standard practice much of the time (§5.2), even when using more complex initial surface methods like those discussed below.

Particle set to surface: Rather than simply using independent radial functions to define an iso-surface, other works have looked to use information about the actual distribution of the particles as well to improve the smoothness (and sharpness where required) of the resulting surfaces. Zhu and Bridson [2005] propose a method that creates temporary particles with locally-averaged positions and radii to create smoother surfaces for their FLIP/PIC simulations. This uses a minimum distance metric rather than an accumulated density such that it can handle the non-uniform particle distribution of these simulations, but the averaging and temporary 'particles' used help to create smoother surfaces than simple sphere stamping. This approach is known to create some artefacts in concave regions and edges that were later tackled by the extension of Solenthaler and Pajarola [2008] by using further information about the particle distribution. However this more advanced method requires use of unintuitive eigenvalue thresholds to do so. A further extension of this method is shown by Adams et al. [2007] who use re-distancing to re-sample the positions of the points and then create a surface using the method of Zhu and Bridson [2005]. This method is shown to create smooth surfaces even with varying radii particles but requires knowledge of previous timesteps to perform this re-distancing and so is not as readily applicable purely as a post-process.

Yu and Turk [2013] consider an extension to per-particle methods like Blinn [1982] to use anisotropic kernels, deforming the footprint of each particle dependent on its local particle distribution. In this way, smooth surfaces and thin-sheets are better captured as kernels are stretched to join neighbouring particles. Whilst the original work of Yu and Turk [2013] use a density-style scalar field (retaining Blinn [1982]'s issues with thickening etc.), Ando et al. [2012] simplify this to instead use a distance function, stamping fixed-radii ellipsoids into a scalar field (analogous to the spherical method of Museth [2014]), noting that this only requires very minor post-processing on the subsequent mesh to create a renderable result. Our work extends this approach to be more robust to varying particle distributions and work in the general case of using particle simulation data from different kinds of simulation e.g. SPH, FLIP/PIC etc.

Motivated by their adaptive particle simulation method, Ando et al. [2013] develop a method that is robust to varying particle radii, which other methods often struggle with. This works by finding sets of 3 particles (possibly of varying radii) and creating convex hulls that encapsulate the spheres around them, unioning these convex hulls together to create the complete liquid surface. However, as this method is closely tied to their adaptive simulation, relies on their BCC mesh structure and is shown to be computationally expensive, it may not be worthwhile for the case of more common, uniform radii simulations.

Whilst these methods attempt to capture as much detail as possible from the simulated particle set, there are also works that focus on increasing the detail beyond the original particle set or surface.

Surface up-resing: Instead of trying to maximise the surface detail directly out of the particle-to-surface operation, there are have been works that instead look to take a surface and infer further details beyond those that come from the initial simulation.

Methods such as Kim et al. [2013] and Mercier et al. [2015] define approaches to create enhanced details on liquid surfaces, adding finer scale wave dynamics on top of existing simulations. The method and results demonstrated in Kim et al. [2013] takes their input surface from a level-set liquid solver but the possibility of extending their approach to use a surface generated as a post-process from a particle set (via one of the above methods), is discussed. Mercier et al. [2015] describe their method solely from an input particle simulation which could allow the possible application to our target of working with general particle simulation data. These methods are both shown to successfully increase surface details for wave and ripple behaviours which could be very useful for large bodies of liquid, but do not allow improvement to splashing and violent liquid motions as they can not 'up-res' isolated droplet behaviours. Other surface displacement approaches have also been used in production[Budsberg et al. 2013] to create similar finer-scale details.

These methods could be beneficial in the future (and some may be compatible with our work) but the rest of this chapter will focus on initial surface creation, and in particular how best to create details in violent and splashing liquid phenomena, a common FX production problem.

5.4 Our Approach

The approach we take is based on the work of Yu and Turk [2013]. Applying their local-neighbourhood dependent deformation of usual isotropic kernels we create smooth surfaces at the interface between particles and air, as well as achieve good handling of thin sheets and splashes. For droplets, as in Yu and Turk [2013], we find isolated particles during these calculations to process differently, choosing to retain isotropic kernels for these under-resolved particles, in turn recreating normal spherical droplet shapes. However, we make two modifications to this method:

- Use of a distance function for the initial isotropic kernel (Equation 5.2, as in Ando et al. [2012]) for robustness to uneven particle distributions and clumping (a common occurrence in FLIP simulations - Chapter 2).
- Better use of geometric information in calculation of these deformations, removing an unintuitive and potentially problematic user parameter involved in the scaling of the anisotropic kernel (Equation 5.7).

5.4.1 Robust Anisotropic Kernels for Particle Fluid Surfacing

Consider our liquid surface to be defined by the iso-surface of the scalar field $\phi(\mathbf{x})$. Supposing we define a simple spherical distance function per-particle, this final field would be given by:

$$\phi(\mathbf{x}) = \min_p(||\mathbf{x}_p - \mathbf{x}|| - r_p) \tag{5.1}$$

where $\mathbf{x}_p$ is the position, and r_p the radius, of particle p.

Following the anisotropic kernel approaches of Yu and Turk [2013] and Ando et al. [2012] we look to find a deformation matrix G_p s.t. we can define our scalar field as the ellipsoid distance function:

$$\phi_G(\mathbf{x}) = \min_p(||G_p(\mathbf{x}_p - \mathbf{x})|| - r_p) \tag{5.2}$$

Calculating the anisotopy

The deformation matrix G is calculated using information of the neighbouring particles and Weighted Principle Component Analysis (WPCA) as in Yu and Turk [2013]. That is, we calculate the weighted mean $\bar{\mathbf{x}}_p$ and covariance matrix C_p for each particle as:

$$\bar{\mathbf{x}}_p = \frac{\sum_i w_{ip}\mathbf{x}_i}{\sum_i w_{ip}} \tag{5.3}$$

$$C_p = \frac{\sum_i w_{ip}(\mathbf{x}_i - \bar{\mathbf{x}}_p)(\mathbf{x}_i - \bar{\mathbf{x}}_p)^T}{\sum_i w_{ip}} \tag{5.4}$$

where the surrounding particles i have positions $\mathbf{x}_i$ and weights w_{ip} are given by the isotropic function:

$$w_{ip} = \begin{cases} 1 - \left(\frac{||\mathbf{x}_i - \mathbf{x}_p||}{R}\right)^3 & \text{if } ||\mathbf{x}_i - \mathbf{x}_p|| < R \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

where R is the search radius defining the neighbourhood.

Now given the matrix C_p, we take the SVD (here equivalent to the eigenvalue decomposition as C_p is symmetric, positive semi-definite) to get the eigenvalues and eigenvectors as:

$$C_p = R\Sigma R^T \tag{5.6}$$

where R is a rotation matrix ($\in SO(3)$) and Σ is a diagonal matrix of the eigenvalues (σ_0,σ_1,σ_2) descending in size.

Ellipsoid eigenvalue manipulation

These eigenvalues represent the stretches of the deformation matrix G_p. To avoid deforming our kernels too aggressively (i.e. a deforming a 3D sphere to a 2D ellipse), we clamp them such that they do not differ by too great an amount, i.e. $\hat{\sigma}_i = \max(\sigma_i, k_r\sigma_0)$ for $i = 1, 2$. Similarly, to prevent poorly defined (or rapidly changing) deformations due to incomplete particle neighbourhoods (i.e. number of particles n within search radius R s.t. $n \leq n_{droplet}$) we can also change behaviour here and ignore the stretches, replacing Σ with the identity matrix scaled by some parameter $k_n < 1$ (to create smaller spherical 'droplets'). Furthermore, the stretches in Σ vary dependent on the local distribution but do not necessarily map to a volume preserving transformation, so in Yu and Turk [2013] they incorporate a user parameter k_s here to 'normalise' these stretches i.e. try to set $||k_s C_p|| \approx 1$. *However*, whilst recognising this issue and suggesting this solution they later state that they use a single value of $k_s = 1400$ throughout their examples and offer no solution for handling the calculation of this value, as such leaving the value as a parameter for the user. This value controls the eventual magnitude of the ellipsoid (or kernel) radii so having the correct value is very important to creating the final surface. We can see in the example in Figure 5.1 that the ideal value of this parameter varies dependent on the input data *and even* per-particle within this data.

Our solution to this uses simple properties of the deformation matrix C_p, or in particular the eigenvalues σ_i. As the volume change given by a deformation matrix (in our case C_p) is equivalent to scaling by the determinant of that matrix (as $R \in SO(3)$, $det(R) = 1$, therefore $det(C_p)$ is the product of the diagonal values of Σ) we can calculate a volume-preserving scale to apply to all of the stretches to get k_s as:

$$k_s = \begin{cases} \frac{1}{det(C_p)^{\frac{1}{3}}} = \frac{1}{(\sigma_0\sigma_1\sigma_2)^{\frac{1}{3}}}, \text{ if } \sigma_i \neq 0 \text{ for all } i = 1,2,3 \\ 1, \text{ otherwise} \end{cases} \tag{5.7}$$

An alternative way of calculating this value that we also have considered (more akin to the user choosing a single value as in Yu and Turk [2013]) is to calculate the average volume change for all particles $p = 1, ..., P$ say $\bar{v} = \frac{\sum_p det(C_p)}{P}$ and let $k_s = \bar{v}^{-1/3}$. These two approaches allow volume preservation on either a local (per-particle) or global (all particles) level. In practice we allow a user to blend between these two methods. It should be noted that of these methods, Equation 5.7 is most robust as the calculation is per-particle and so guarantees reasonable values (for all reasonable inputs) unlike the averaging or 'global' method can be sensitive to large variations in $det(C_p)$, i.e. variation in the distribution of the local neighbourhood (although these large variations are mitigated by using the 'droplet' isolation).

To construct the final deformation the modified stretches are then inserted into the matrix C_p by replacing the matrix Σ with $\hat{\Sigma}$ where:

$$\hat{\Sigma} = \begin{cases} k_s \text{diag}(\sigma_0, \hat{\sigma}_1, \hat{\sigma}_2) & n > n_{droplet} \\ k_n \mathbf{I} & \text{otherwise} \end{cases} \tag{5.8}$$

Resulting in:

$$\hat{C_p} = R\hat{\Sigma}R^T \tag{5.9}$$

Then this is inverted to get $G_p = \hat{C_p}^{-1}$ (by replacing $\hat{\Sigma}$ with its inverse) and applied to the isotropic distance function as in Equation 5.2.

5.5 Implementation

This has been incorporated into a particle surfacing node for Houdini that also allows artists to create surfaces using simple spherical distances (as in Equation 5.1) and using the method of Solenthaler and Pajarola [2008]. The node uses VDB for the storage and iteration over both volume and point data. As the deformation calculation requires comparatively expensive point neighbourhood lookups, users are able to select subsets of their input point sets upon which to calculate the ellipsoid deformations to avoid unnecessary computation in regions far beneath the surface. Yu and Turk [2013] suggest a similar method for optimisation, but their approach requires calculation of points to skip using the costly neighbourhood lookup that we wish to avoid, instead we propose users consider other geometric approaches such as using the VDB points grid topology to determine these areas.

The node takes a VDB points particle set and calculates the deformations dependent on its surrounding neighbourhood (whose width is defined as a user parameter as a scale on the isotropic particle radius). The output of this operation is a signed distance field VDB that represents the distance to the surface created by the overlapping the ellipsoids. Users are then able to post-process this as they like, using all of the available VDB tools e.g. level-set filtering (or convert to a mesh and use mesh processing tools).

We also include the ability to perform the positional smoothing of the ellipsoid centres as suggested by Yu and Turk [2013] which we similarly find to be very important to the overall quality of using this method.

One element of liquid surfacing methods that is often overlooked outside of production is the transferral of other information such as velocity (critical for motion blur). As we reconstruct the surface on each frame there is no explicit temporal coherence, especially if we convert our signed distance function to a mesh representation, so we are unable to calculate this from the final surface representation. Thankfully, the information for velocity does exist on the particles so we can use a similar method to transfer the velocity as we do for stamping the ellipsoid. In this way, we stamp the velocity value of the 'closest' (in the ellipsoid deformed space) point into the volume at each voxel as we do for the distance value.

5.6 Results

We demonstrate the use of this method on some of DNEG's liquid simulation training data in Figure 5.4. Here the splash details and droplets are retained whilst creating particularly smooth surfaces in the splash itself and relatively smooth surfaces in the surrounding ocean. The simulation data here comes from a FLIP simulation and so has no hard constraints on the distribution of the particles. The texture of the ocean surface comes from the simulated data where smoothing quality of the ellipsoid deformations is unable to smooth out the larger-scale features in the particle set. This could easily be removed by a single masked smoothing pass, whilst the highly detailed, yet smooth, splash requires no further processing.

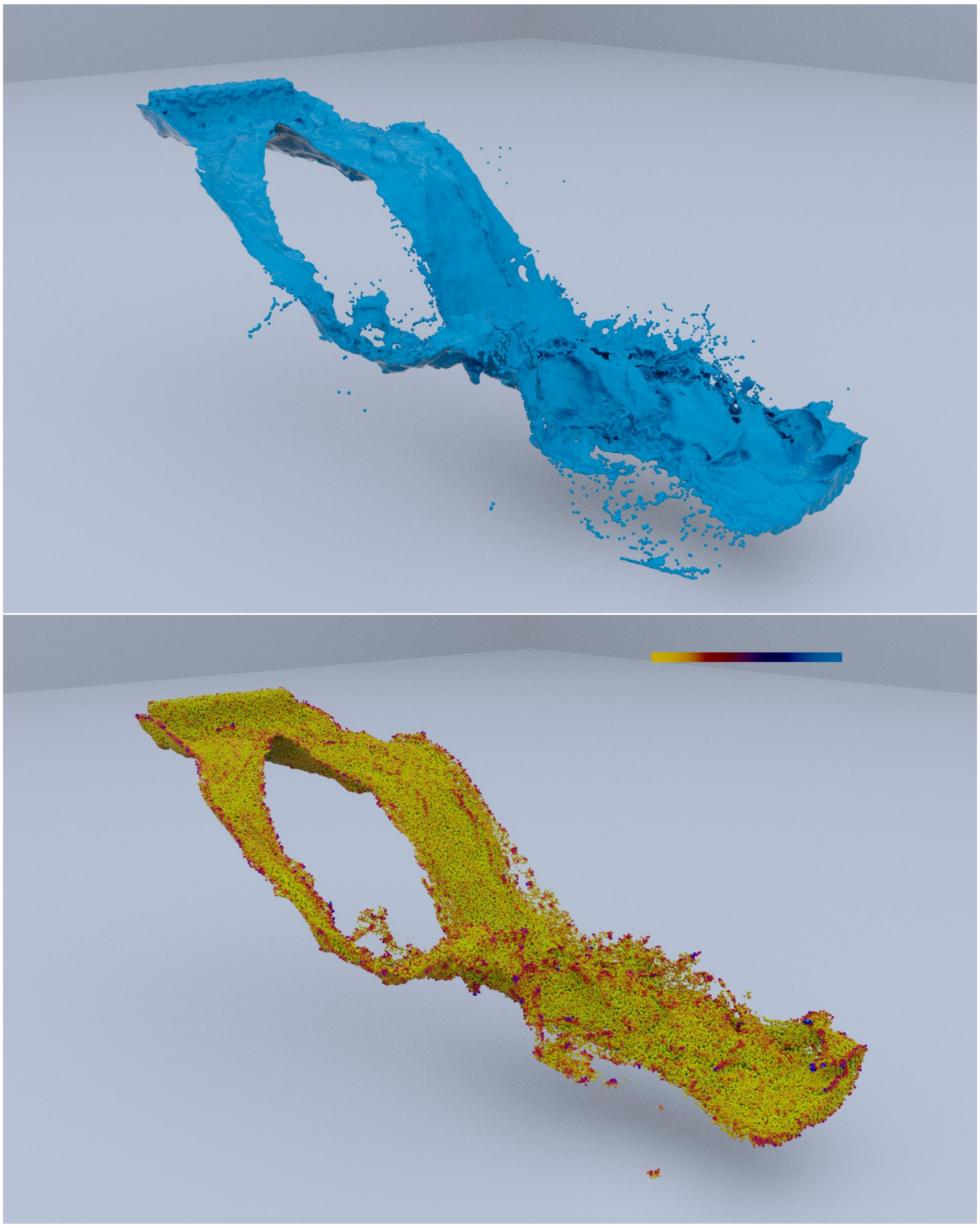


Figure 5.1: Results of a FLIP liquid simulation. a) Final surfaced result using our method. b) Particles coloured by their calculated value for k_s using Equation 5.7, values vary from 114—553 (droplets removed, color map: 144 - yellow to 553 - blue). Variation in this parameter occurs at the surface and at areas of interest where droplets break off from the main liquid body.

In Figure 5.2c, we show the difference between the use of 'local' or 'global' values for the deformation normalisation. The boundary particles near the interface generally have a smaller imprint in the 'global' case as the inner regions of the surface bring the average down to below what would be required for boundary particle volume preservation. It should also be noted that the automatically calculated value of the single 'global' normalisation constant is 152, whereas the suggested analogous, user-defined value from Yu and Turk [2013] is 1400. If their value were used the surface would be inflated, stamping particles of 9.2x the size of particle we calculated, leading to undesirable results (Figure 5.2cc).

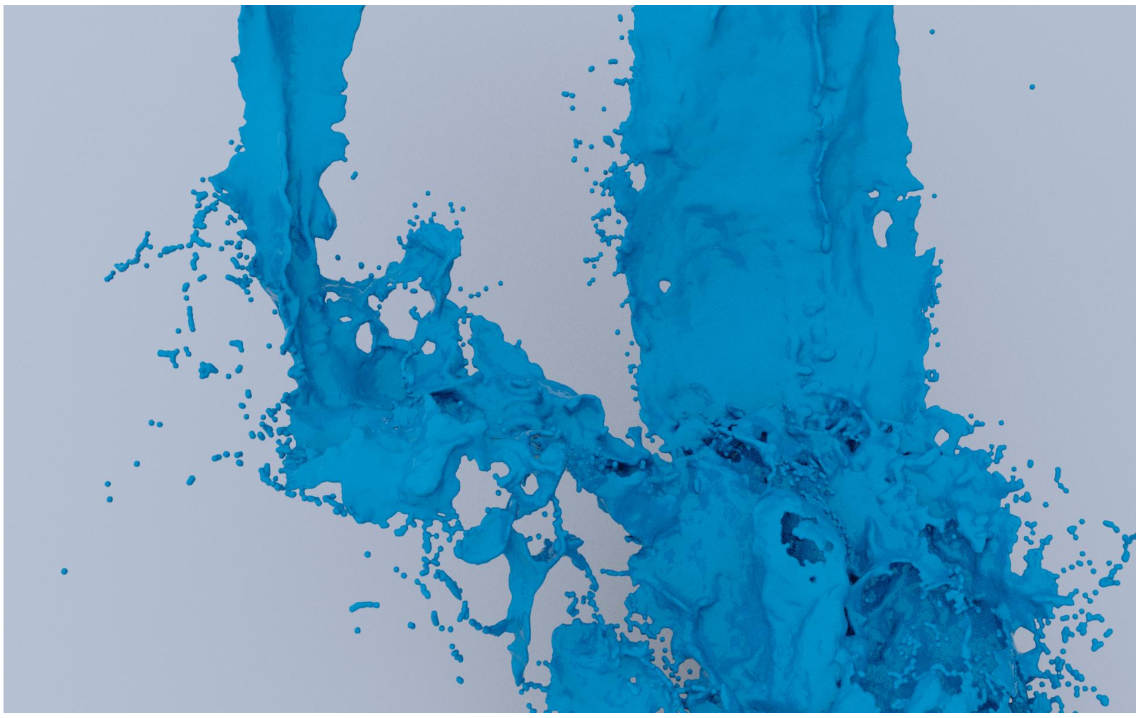
We also include statistics of use of the Houdini node in production at DNEG in Figure 5.5. It should be noted that whilst these statistics cover all three modes of operation (i.e. spherical, ellipsoid or Solenthaler and Pajarola [2008] methods) of the node, direct feedback from the artists suggests that the method described in this chapter is the preferred approach and most likely large proportion of these uses, especially as the other methods are currently without velocity transfer implementations.

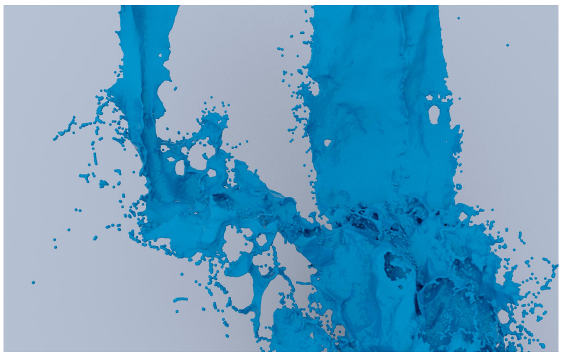
5.7 Analysis

Use of the distance function to define the surface from Ando et al. [2012] rather than an iso-surface from the density field as in Yu and Turk [2013] also allows us to take advantage of storing only a narrow band level-set [Museth 2013] rather than a dense field. This helps to massively reduce the memory footprint of the resulting scalar field when using a sparse volume structure like VDB as only a thin band of voxel values need to be stored. We also found that the distance function was more robust than the density function to the varying particle distributions (and in particular clumping) that occurs in simulation frameworks such as FLIP that do not enforce the particles to be well distributed.

The key contribution of our work are the methods for determination of the deformation normalisation parameter k_s. In the work of Yu and Turk [2013] this is given by a seemingly arbitrary number with very little intuition behind if it were exposed to a user. We have also demonstrated that if left as a single user parameter this could be destructive in practice, as values determine the scale of the resulting ellipsoid radii. In areas where this method has most effect (where the ellipsoid deformations should occur), i.e. near the air-liquid interface, the distribution of the particles can vary significantly (Figure 5.1) therefore if using a fixed normalisation constant so too would the volume of each created ellipsoid. This could potentially create massive or tiny ellipsoids which could cause issues ranging from crashes from running out of memory to seemingly giving no result at all. In our updated approach the ellipsoids are guaranteed to be of reasonable size and shape by the combination of both the user defined k_r enforcing 'sphericity' and the new automatically and adaptively calculated k_s enforcing volume preservation.

As this method requires neighbourhood lookups to calculate the deformation and an SVD per-point, it is more expensive than simple spherical stamping (Equation 5.1). Methods such as only calculating these only for a subset of input points help to alleviate the issue but currently require artist input so would be good to automate in the future. Similarly the control of the droplet shape and size is quite simple and it may be interesting to look to improve this with


(a) 'Local' ellipsoid normalisation. Values for normalisation scale k_s vary between $114 - 553$.


(b) 'Global' ellipsoid normalisation. Value for normalisation scale k_s calculated by averaging gives $k_s = 152$.

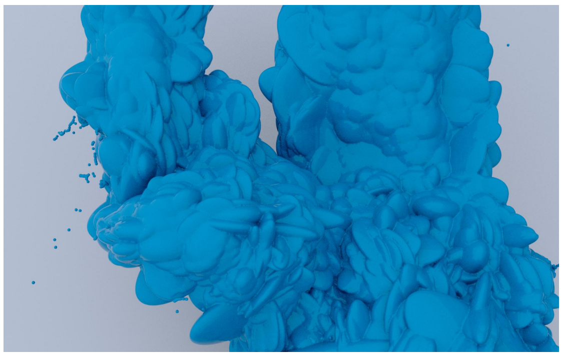
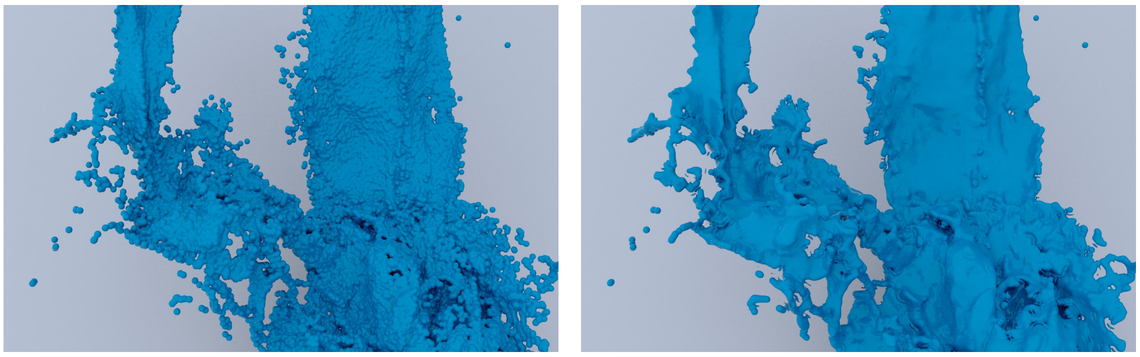

(c) User defined ellipsoid normalisation, using default value suggested by Yu and Turk [2013], $k_s = 1400$.

Figure 5.2: Comparison of surfaces with different ellipsoid normalisation values. The 'local' method (a) better preserves sheets at the edges whereas 'global' (b) can shrink the particle imprints at the edges to produce thinner sheets more prone to holes. As the values for this are dependent on the input particle set and other parameters of the model, i.e. search radius R, using the wrong value can lead to undesirable results as we see in c).



(a) Surfacing using an isotropic distance function (Equation 5.1). This is the starting point for the workflow of from Museth [2014] but the immediate result is not of renderable quality.

(b) Surfacing using the method of Solenthaler and Pajarola [2008]. This displays known artefacts at points where droplets break off from the main body, which are possible to remove but require fine tuning of unintuitive eigenvalue thresholds.

Figure 5.3: Alternative methods for particle surfacing. Both the bumpy surface of the isotropic distance function and the artefacts on the method of Solenthaler and Pajarola [2008] would require post-processing to create a suitable liquid surface. Even then these would fail to capture fine details such as thin sheets and droplets as well as the method we describe.

say velocity trails [Museth et al. 2007] or a more sophisticated determination of size (e.g. a blending method using the number of neighbouring particles).

Even with its performance trade-off, we have seen great adoption of this method at DNEG, demonstrated in the usage statistics of Figure 5.5. It should be noted that many other methods remain available to the artists, particularly those allowing creation of particle surfaces using the other methods demonstrated in Figure 5.3b (quicker to create an initial surface) - this suggests that the improved levels-of-detail and smooth surfaces captured immediately by our method are preferred over the usual post-processing reliant workflow. From these statistics we can also see that one show dominates the usage in the time-frame considered, this is the film *Venom*. A large part of the FX work on this film involves an organic 'symbiote' and tentacle-like effects, many of which have been modelled using bespoke particle systems. It has become clear that this improved method for creating surfaces from particle systems has application in the FX tool-set beyond only liquid simulation.

Working with this tool in production on "bubbling lava splashes" in *The Kid Who Would Be King*, one artist commented on how it was able to avoid "flickering artefacts" that commonly occur in previous approaches, stating:

"This worked amazingly well on the bubble areas that traditionally would see flickering artefacts as the fluid separated from the main body." - Andy Guest, FX Artist.

5.8 Summary

In this chapter we have introduced and improved upon a state-of-the-art method for creating liquid surfaces from particles. By combining the use of a simple minimum distance implicit

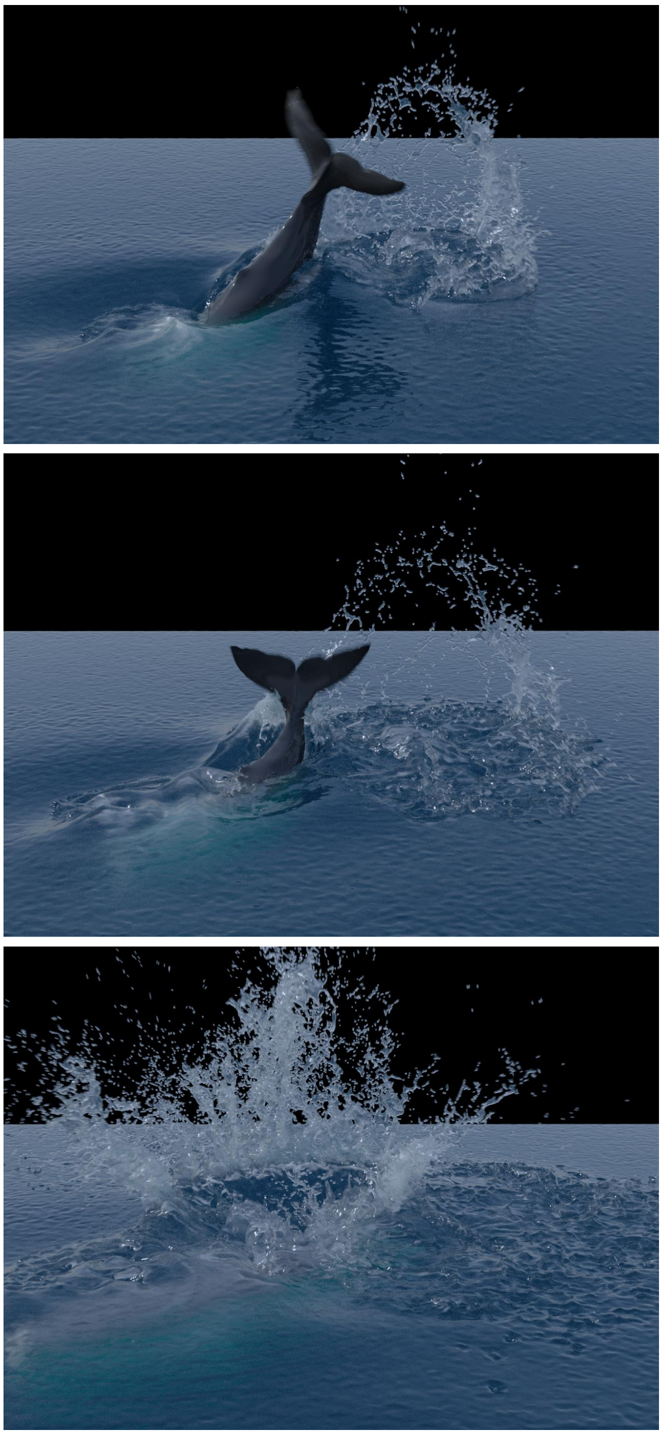
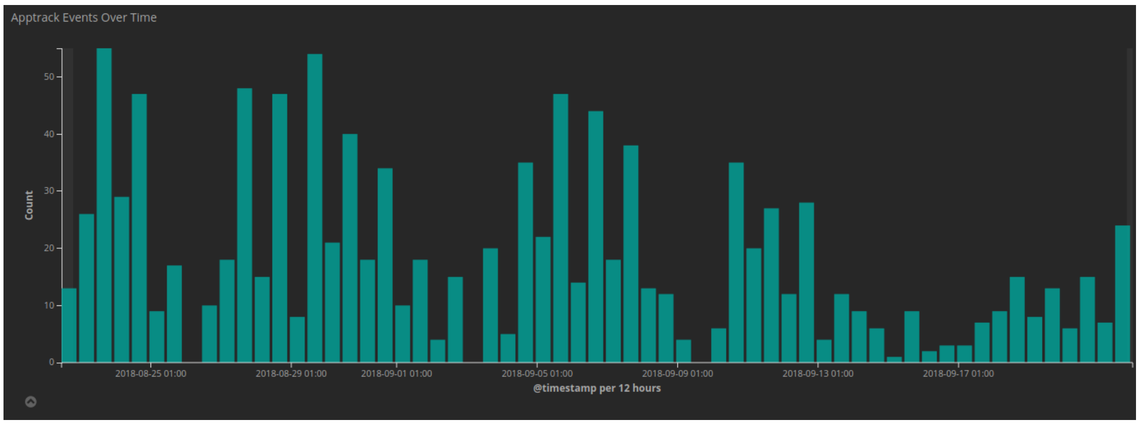
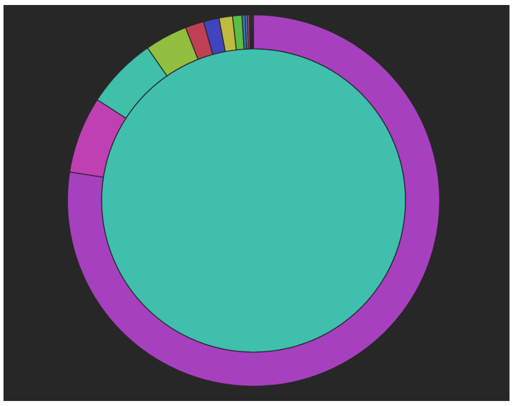


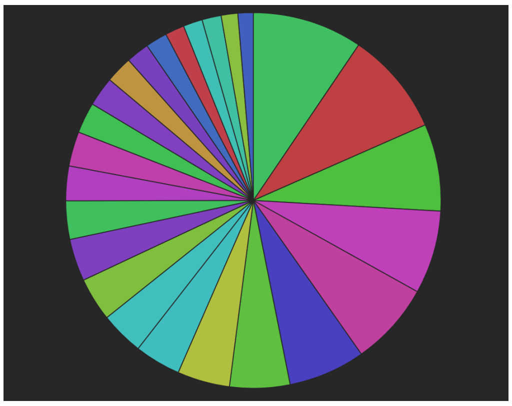
Figure 5.4: A whale splash simulation taken from DNEG's training material, surfaced using the method described in this chapter. Note the small droplet details and smooth surfaces, especially in the splash.



(a) Uses per day. Total uses: 1109.



(b) Uses broken down by show. Number of shows this has been used on: 15. Size of outer ring segment represents proportion of uses.



(c) Uses broken down by user. Number of unique users: 80. Size of segment represents proportion of uses.

Figure 5.5: Statistics for uses of the described Point Surfacer node in Houdini at DNEG, for 30 days prior to September 21st 2018. Graphs taken directly from Kibana.

function for our particles (thus allowing use of a narrow-band level set) with a deformation driven by the local particle distribution and our improved normalising factor based on volume preservation, we have been able to create a robust, production-ready method that creates smooth surfaces whilst retaining fine details. This approach has become the standard for particle surfacing at DNEG, seeing extremely heavy use since its introduction (Figure 5.5), including use on at least 15 different films in production by 80 unique users.

Future work in this area could look to speed-up computation, or, increase user control through better utilisation of other data such as number of neighbours or particle velocity. Furthermore, better coupling and transitions between liquid surface simulations and other secondary simulations such as droplets (Chapter 4) remains an area of possible improvement.

Chapter 6

Increasing Control: Particle and Volume Manipulation with AX

*More so than the previous projects, the following chapter describes a direct collaboration between myself and the FX R&D team at DNEG. This is a project to which I am a key contributor, developing throughout the course of the research project, up to and beyond its open-source release in August 2018. The initial development of an expression interface for changing point data was a combined effort between all authors of the accompanying paper [Avramoussis et al. 2018] (myself included) whilst the extension of this framework to volume data was that of other developers at DNEG. My contribution has had particular focus on applications of this tool to particle simulations, for example the ability to move points. The majority of the content of this chapter was presented in the paper **A JIT Expression Language for Fast Manipulation of VDB Points and Volumes** at DigiPro 2018. This tool was used heavily in the development of the models of Chapters 4-5 as well as in production on the Academy-Award winning Blade Runner 2049 [Villeneuve 2017] and other films including Pacific Rim: Uprising [DeKnight 2018] and Godzilla: King of the Monsters [Dougherty 2019].*

6.1 Overview

This chapter presents the development of an expression language, AX, to allow artistic control of geometric data in a flexible and user-friendly way, specifically focused on promoting reusable and transferrable interactions throughout the VFX pipeline. This is demonstrated through application to point and volume data stored in VDBs. Whilst the other contributions of this research project focus on improving on the underlying simulation systems being used in FX production, this work instead looks to offer greater freedom to artists and tools developers alike, such that they may implement changes to simulation systems themselves and further facilitate other creative, experimental and bespoke workflows as well (which are critical to practical VFX production, as we have seen in Chapter 2).

6.2 Motivation

Modern FX artist workflows are extremely varied and complex. Whilst many of the core technologies used (such as simulation frameworks, i.e. FLIP/SPH) have not changed a great deal in recent years, the other tools around them and in particular the level of control they expose has steadily improved. DCCs such as Houdini now offer both pre-packaged workflows for larger simulation setups, whereby an artist can press a button and generate a 3D FLIP simulation guided by the motion of a 2D heightfield ocean whilst interacting with pre-generated beach geometry, as well as the all of the fine-grained components that they are composed of. Artists are able to dive in to every stage of these systems, adding new operations and/or making other wholesale changes. With this kind of tooling available, such flexibility has rapidly became an expectation/requirement of the modern artistic workflow.

Meanwhile, open-source technologies have become hugely popular and are often depended-upon to provide both reliable and efficient representations of data, and facilitate its subsequent transfer throughout the pipeline. The successes of Alembic [Sony Pictures Imageworks 2018a], OpenVDB [Dreamworks Animation 2018], USD [Pixar Animation Studios 2018] and other open-source geometry and scene representation formats are great examples of this. Their open-source nature allows integration into almost any parent application (of which there are many [Side Effects 2018; Autodesk 2018; Isotropix 2018; Foundry 2018]) to cater to the ever-changing needs of a modern VFX studio.

Whilst these two characteristics of VFX pipeline are not diametrically opposed, their combination can lead to some issues. Exposure of the expected level of functionality for these open-source technologies in all of the applications that support them, requires a huge amount of development. Further still, different applications may not accommodate the same workflows, putting further strain on tool developers to implement extra application specific logic. An obvious solution to these problems is to only expose certain functionality within certain applications, where it is deemed most appropriate, and rely on artists using different applications as required. In reality however, artists are often only familiar with a select number of these applications and to make required changes may mean sending work back to other artists in different stages of the pipeline, which can be costly and disruptive.

Considering the benefits of using these open-source technologies and the familiarity of data-representation that they provide, a similarly unified way to allow manipulation of this data would be a great tool to take advantage of this portability, and could alleviate many of the issues with duplicated and varying functionality. Taking OpenVDB as an example, interaction with the data is currently provided only in the form of either the extensive C++ API, or the very limited Python API. This means that much functionality is only available to tool developers (likely knowing C++) who are required to expose this for artists (who are likely to have some Python experience). The development overhead of creating C++ plugins means that it is unlikely that one-off or bespoke tools are feasible in practice, even for artists with the required development experience. Tools such as Houdini's VEX (a shader language that allows geometry manipulation) [Side Effects 2018] demonstrate a good alternative to this, offering a specialised interface for working with FX data that is both fast to write and execute. However, as part of Houdini, VEX is a proprietary language and whilst it offers a lot of ability to develop new functionality within Houdini, is not hugely flexible outside of the application or its native datatypes. Even supported open technologies like VDB volumes requires sacrificing efficiency (relying on intermediate conversion between VDB and its own volume storage[1]), and there is no way for external parties to add support for other data such as VDB points. Most importantly, as a closed technology it is not truly portable and so is unable to be the unifying interface that we require. Its successes do however demonstrate the usefulness of an efficient, intuitive programming interface for FX data and how such a thing could be a hugely beneficial tool to be used by FX artists and tool developers alike.

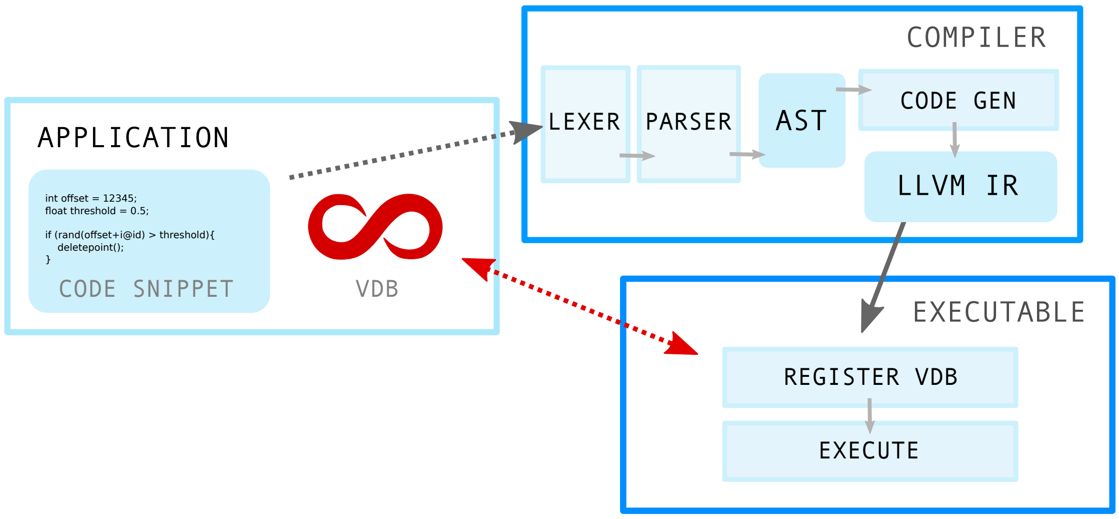
6.3 Portable and Domain-Specific Languages

The integration of general purpose languages, particularly scripting languages such as Python, Lua etc., into DCCs offers users of different applications a familiar programming interface, whilst also giving custom application and task specific functionality. However, these languages are commonly unable to provide the performance requirements to operate on heavy data structures such as representations of geometry used in FX. A custom operation on billions of points, vertices or voxels will likely require careful algorithmic design and a good multi-threaded framework to achieve practical results. Such tasks instead tend towards compiled methods such as C++ plugins which can to have issues with portability and complexity (developer-facing) and can only expose limited functionality (user-facing). Even though languages such as Python can often be extended with C++, exposing the ability to perform custom operations on these non-standard datasets (i.e. not simple arrays) would most likely require a significant amount of engineering by the user.

One way to overcome these limitations is the use of domain-specific languages, whereby the language is designed for application to specific problems, and so its architecture and design can be more focused and succinct. Recent works in graphics include languages for simulation [Bernstein et al. 2016; Kjolstad et al. 2016; DeVito et al. 2011] as well as a large number of other works in applications for rendering and shading (often known as *shader* languages) [Hanrahan and Lawson 1990; Segal and Akeley 1999; Blythe 2006; Mark et al. 2003; Parker et al. 2010]. These shader languages are particularly successful when coupled with Just-In-Time (JIT) compilation, which can allow them to be written inside of other applications e.g OSL [Sony Pictures Imageworks 2018b], VEX. These are often exposed much like scripting interfaces, but their targeted design and (usual) compilation results in far more efficient execution, making them suitable for computationally demanding tasks such as ray-tracing or even geometry manipulation. As they follow specific execution patterns or only consider particular functionality, these languages are generally very simple to write and avoid a lot of boilerplate code, making them a great candidate for a flexible user-facing tool. In developing such a language, the heavy-lifting of development is done once on the developer side, to create the framework for function generation and execution, and after that it becomes very quick and easy (for users) to make bespoke operators.

LLVM [LLVM Developer Group 2018], used in developing the clang compiler, gives developers the ability to create their own JIT compiled languages, which are able to offer performance comparable to ahead-of-time compiled languages (like C++). Developing a similar 'shader'-like

[1]Personal communication with OpenVDB developer Ken Museth, 2018



```
int offset = 12345;
float threshold = 0.5;

if (rand(offset+i@id) > threshold){
    deletepoint();
}
```

Figure 6.1: The structure of this tool and how it integrates with a chosen application, broken down into the two main components, the compiler and the executable.

language to manipulate widely supported FX data formats could provide great flexibility for VFX artists, and using LLVM's JIT compilation can allow this to be done without sacrificing performance.

6.4 Our Approach

We have developed a new expression language for the manipulation of FX geometry data (currently implemented for point and voxel data in VDBs), that is:

- Fast - speed comparable to a compiled, multi-threaded plugin
- Portable - transferrable across DCCs and applications
- Easy-to-use - requires minimal programming experience

This has been used to provide greater artistic control throughout the pipeline at DNEG with exposure in Houdini, Clarisse [Isotropix 2018] and from the command line. The flexibility it provides has been particularly useful on recent shows such as Blade Runner 2049 and Pacific Rim: Uprising as part of the in-house scattering and point tool-set.

We will now consider how this expression interface is designed/implemented, some of examples of how it can be used and the impact it has had on production.

6.4.1 Task-Specific and Parallel by Design

Shader languages offer a nice solution to the boilerplate code required with other less task-specific languages, in our case allowing an easy element-centric expression interface that can be efficiently parallelised. Our design focuses on the manipulation of the smallest or lowest element in our geometry (VDB), i.e. a point or voxel, executing the user-supplied expression over each element independently. Taking points for example, these elements often have a variety of 'attributes' such as position, velocity, colour, radius, normal direction, or other arbitrary quantities that they store and carry. Using our element-centric design users can then write kernels that retrieve, use and update each elements values for these attributes to perform a range of operations (§6.6) without requiring initialisation of looping over the points or indexing into stored attribute data. The iteration process over these elements is embarrassingly parallel and when coupled with our highly-optimised, JIT-compiled functions is extremely efficient and capable of handling very large datasets at interactive rates.

6.5 Implementation

The implementation can be broken down into two main components (demonstrated in Figure 6.1): the LLVM function generation that takes an input expression and JIT compiles it into a custom function to be run on each geometric element; and the OpenVDB component that registers access to the supplied VDBs and subsequently performs execution over them. These are implemented as a C++ library using a Flex and Bison lexer/parser for the handling of the grammar. The plugins to Houdini and Clarisse, and our standalone executable are then built separately, dependent on the core OpenVDB AX library.

6.5.1 LLVM Function Generation

The use of LLVM allows us to generate custom compiled functions from our expression language, and comes with built-in support for a number of optimisation and validation passes. As shown in Figure 6.1, we begin by parsing an input code snippet to create an abstract syntax tree (AST), a representation of the syntactic constructs that occur in the supplied code. This is traversed by our code generation framework which converts each node of the AST into LLVM's Intermediate Representation (IR). Finally, the resulting IR is JIT-compiled and handed to our executable for later execution. The parsing, IR generation and JIT-compilation are very fast operations and due to the intensive optimisation passes offered by LLVM, the resulting functions are of comparable speed to ahead-of-time compiled C++ code (Table 6.1). We also make sure to decouple this process from any particular input VDB, therefore only requiring a single compilation pass for any number of inputs. Instead, each unique input simply undergoes a fast registration step before execution to expose the correct data to the compiled function.

6.5.2 OpenVDB Integration

In our expression interface we allow read-and-write access to OpenVDB point attributes and voxel values. Our OpenVDB bindings give our compiled functions direct access to the values within any supplied VDB trees. To do so, on execution we bind handles to point attributes/volumes and store them on our executables. To ensure we only require a single code generation pass for any number of VDBs we do not compile these directly into IR. Instead we insert IR instructions to retrieve these handles at runtime[2].

For multi-threaded execution we leverage OpenVDB's node structure [Museth 2013]. In this way, we parallelise over each node in the supplied VDB trees, executing our function on each element (point/voxel) they contain. This execution pattern fits intuitively into OpenVDB's data structure, inherently load-balancing over its spatially-organised nodes.

6.5.3 Available Operations

Attribute expressions are written in a simple C-like language with many of the usual syntactic constructs e.g. conditionals, function calls, binary operators etc. (Figures 6.2 & 6.3). We incorporate element accesses through a specific identifier, @ (inspired by Houdini's VEX language), to easily differentiate from local variable usage. The function calls we provide can cover a large amount of functionality, from basic mathematical operations, noise and random number generation to element-specific (and geometry-specific) behaviour such as collecting points into groups.

Whilst currently each element only has access to its own attributes/values, it should be possible to extend to allow access to other elements in future. This could allow even more complex operations like smoothing surfaces or accessing nearby points for neighbourhood operations.

6.6 Examples and Use in Production

In the following, we discuss a couple of different use-cases for this tool and demonstrate the simplicity of the code required to perform such tasks. First we consider an example for FX simulation and then, further downstream in the pipeline, manipulation of FX data in Lighting. We will also discuss some more specific production examples and its overall production use.

6.6.1 Example 1: Simulation

The flexibility given by this expression interface can be used to very quickly create operators such as those required in simple particle or volumetric simulations. For example, consider a particle system acting with respect to a collection of independent motions or driven by external influences - we are able to very easily create this kind of operator using our expression interface as in Figure 6.2. As these operators are often executed many times in sequence, performance is extremely important. We show in Table. 6.1, the performance of our JIT-compiled expression matches very closely to that of an equivalent operator written in C++.

6.6.2 Example 2: FX to Lighting

Having the ability to modify geometry in this way is important in areas beyond FX simulation. However, making modifications or tweaks to geometry is not always easy outside of packages

[2]A similar method is used for other arbitrary external data e.g. time or frame number, allowing use of variable data without requiring re-compilation.

```
// get timestep
float dt = 1.0f / (4.0f * 24.0f);
// gravity
vec3f gravity = {0.0f, -9.81f, 0.0f};
// drag
vec3f dV = {2.0f, 0.0f, 0.0f} - vec3f@v;
float lengthV = length(dV);

float Re = lengthV * float@rad / 1.225f;
float C = 0.0f;
if (Re > 1000.0f) C = 24.0f / Re;
else C = 0.424f;
// calculate drag force
vec3f drag = 0.5f * 1.2f *
        C * lengthV * deltaV * 4.0f * 3.14f *
        pow(@rad, 2.0f);
// update velocity
vec3f@v += (gravity -
    drag / ((4.0f / 3.0f)
    * 3.14f * pow(float@rad, 3.0f))) * dt;
// update position
vec3f@P += v@v * dt;
```

Figure 6.2: A particle simulation step using gravity and drag against a constant wind force.

```
int offset = 12345;
float threshold = 0.5;
// remove points
if (rand(offset+int@id) > threshold) {
    deletepoint();
}
```

Figure 6.3: Decimating points in a set using a percentage threshold.

designed explicitly for it, for example in rendering applications such as Clarisse. Integrating our expression interface into such applications facilitates on-the-fly modifications without requiring costly back-and-forth between FX artists and lighters.

This level of control has seen great use in this context at DNEG, with lighters able to perform many tasks without having to send data back up the pipeline. Some examples we have seen being used include:

- Randomizing scattering ids for instancing.
- Modifying colours, orientation, scales and velocities of scattered point data.
- Decimating excessively large point sets.

Sample code for this latter example can be found in Figure 6.3.

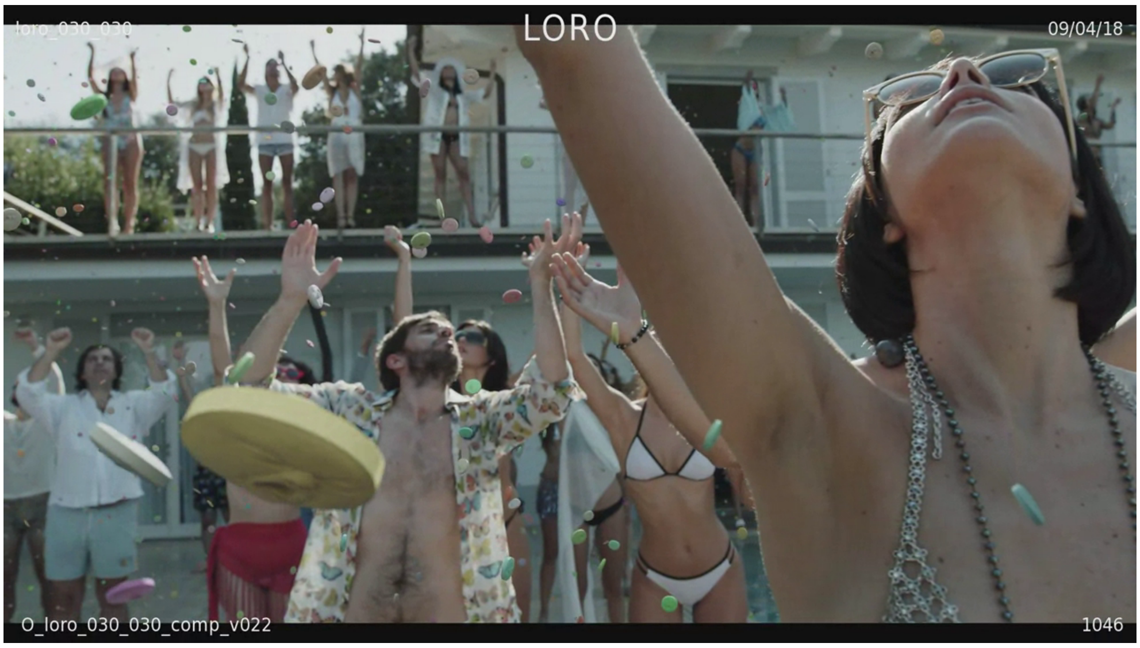
Table 6.1: Performance of C++ implementations vs our JIT-compiled expression examples running on 32 core Intel Xeon 3.10Ghz CPU with 64GB RAM.

Code Example	# elements	C++	JIT	Performance
Figure 6.2	50m points	0.43s	0.58s	0.74x
Figure 6.3	50m points	2.69s	2.70s	0.99x

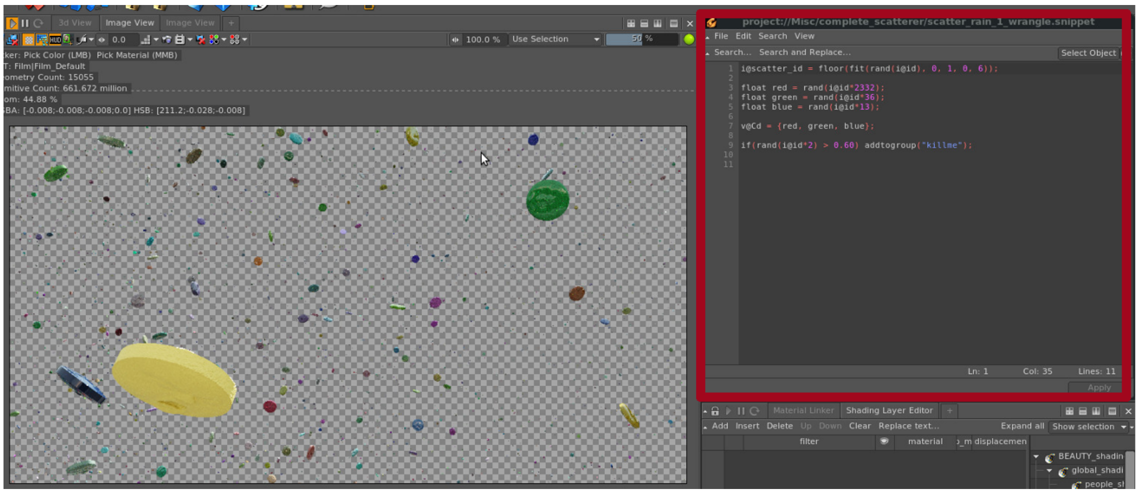
6.6.3 Case Study: Dropping Pills on Loro

The control offered by OpenVDB AX has been particularly useful as part of the geometry scattering toolkit in Clarisse at DNEG. This uses point data to reference and represent geometry such as trees, buildings or anything that requires large numbers of items to be placed around 3D space. This offers a lightweight mechanism for interacting with these items without manipulating geometry (i.e. meshes) directly. The point data can then be later replaced by the scattered geometry if required by the artist, else it will be deferred to only when it is needed, i.e. render time. Attributes on these scattered points can reference different models and are used to define the orientation, scale, colour or any other property that can vary amongst scattered items. By allowing modification of these attributes, and even the position of the points themselves, OpenVDB AX allows artists to author and tweak these representations extremely quickly. By exposing the freedom and functionality of allowing artists to write custom logic i.e. math functions etc. this can allow a huge amount of control that would be otherwise difficult to make into a more conventional, fixed user interface.

An example of this was the use of OpenVDB AX on the film Loro[Sorrentino 2018]. The shots in question required a collection of pills to rain from the sky (Figure 6.4a). Using OpenVDB AX, the artists were able to modify the scattered geometry of the pills by updating the attributes on the points representing them. In this example this meant randomising the colour of the pills (represented by the `vec3f@Cd` attribute) using the `rand()` function with `int@id` (for the random seed), as well as removing a subset of the original scattered points. Other modifications using this setup also included altering the orientation and rotation of the pills through the `vec3f@orient` attribute. These modifications happen in real-time in the Clarisse viewport to allow instant visual feedback and the fast compilation allows artists to modify their code snippets easily and experiment with different expressions.



(a) Scattered pills as they appear in the final shot in Loro[Sorrentino 2018].

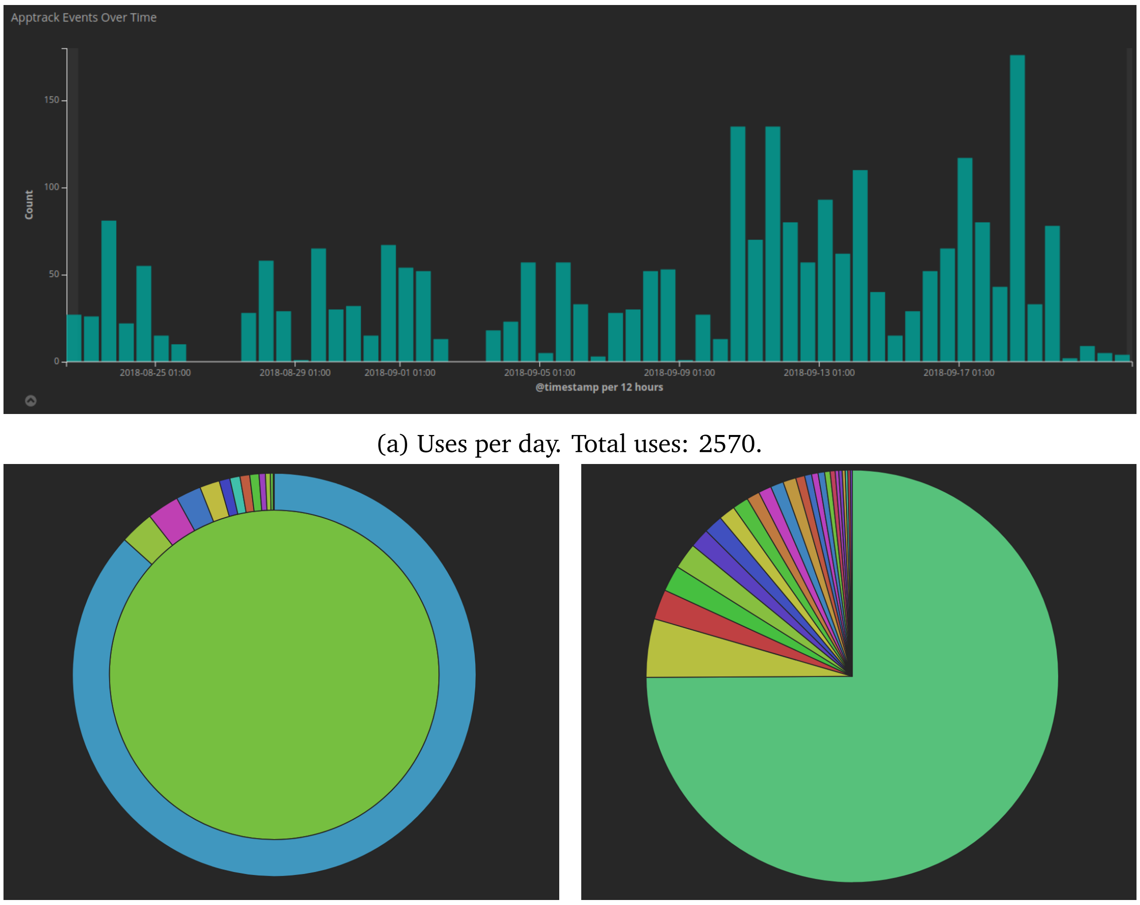


```
i@scatter_id = floor(fit(rand(i@id), 0, 1, 0, 6));

float red = rand(i@id*2332);
float green = rand(i@id*36);
float blue = rand(i@id*13);

v@Cd = {red, green, blue};

if(rand(i@id*2) > 0.60) addtogroup("killme");
```

(b) Using OpenVDB AX in Clarisse to modify the scattered pill geometry. Setting the colour and randomly deleting a subset of the scattered pills. Red outline highlights the code editor used by artists to input OpenVDB AX snippets.

6.6.4 Production Statistics

In Figure 6.5a we can see some internal production statistics from DNEG for use of OpenVDB AX in Clarisse, where it has been most widely adopted. This was used 2570 times over a 30 day period prior to Sept 21th 2018. It has seen use on 13 shows, both for feature film and TV, including Blade Runner 2049 and Pacific Rim: Uprising.



(a) Uses per day. Total uses: 2570.

(b) Uses broken down by show. Number of shows this has been used on: 13. Size of outer ring segment represents proportional number of uses.

(c) Uses broken down by user. Number of unique users: 39. Size of segment represents proportional number of uses.

Figure 6.5: Statistics for uses of the OpenVDB AX node in Clarisse at DNEG, for 30 days prior to September 21st 2018.

6.7 Summary

The motivation for this project arose during the development of OpenVDB Points at DNEG [Museth et al. 2015]. Although originally designed as a particle framework for simulation, it was primarily used for data interchange due to a lack of frontend tooling. The flexibility offered by alternative point formats (e.g. in third-party DCCs) with more comprehensive tool-sets raised further concerns over adoption on a wider scale. To succeed as a toolkit that users could directly interact with, a fast way to expose a lot of custom and controllable functionality became critical. Following previous successes with LLVM [Bailey et al. 2011] and recognising the flexibility arising from geometry shader languages such as VEX, this tool was conceived as a solution to this problem. Once this custom expression interface was exposed for OpenVDB Points, we saw a huge increase in interest in the format as a whole. Then, due to the nature of the integration into OpenVDB, and the similarities between its volume and point storage, it was relatively straightforward to extend this framework to provide manipulation of volumetric data as well. VDBs are now the standard for storage of volumetric and point data at DNEG, with the ability to manipulate them directly and deterministically throughout the pipeline proving to be a significant asset.

The functionality available in the current iteration of this tool is relatively limited (only able to access attribute values for each element independently), but in the future this could be extended to handle access to the rest of the geometry, i.e. neighbourhood lookups or sampling a volume at an arbitrary position. As it is now an open-source project, and has sparked interest from the wider OpenVDB community, we expect further work to build upon this basis in the future.

It should be noted that this language offers functionality similar to VEX in Houdini, albeit in its current form, only a subset thereof. However, there are no other such tools that offer the freedom to manipulate point data in VDBs. Similarly, due to the native access to VDB data, it outperforms Houdini's VEX for volume operations on VDBs which whilst possible are understood to require translation to-and-from its native volume storage format. Most importantly, as this is built upon a transferrable technology in OpenVDB, it gains similar portability and provides a unified way for artists throughout the pipeline to interact with this data.

This tool has shown to have production impact beyond original expectations. We have found that by exposing this interface in multiple applications (e.g. Houdini, Clarisse, command-line) we have been able to speed up both user and developer workflows, allowing on-the-fly edits to assets and exposing access to the data in ways that can also be used in the creation of more advanced tools. Whilst the language and the design itself follow other tools before it, the portability arising from pairing to an open-source technology such as OpenVDB has been invaluable to its success. This further demonstrates the importance of transferrable technologies in a VFX pipeline built upon so many different applications. Interestingly, the decoupled nature of the compiler and execution also suggests that such a tool could in the future be extended to support other types of geometry beyond VDBs such as textures or meshes (as long as we are able to iterate over 'elements' and access their 'attributes').

Chapter 7

Conclusion & Future Work

7.1 Discussion

The positive response of artists to the developments in this thesis suggest that whilst the existing workflows for creating production quality liquid effects are heavily based on artistic input, this is borne out of necessity more so than choice. Looking back to our description of a liquid effect in §2.5 we can see that many of the decisions, such as those to modify the liquid surface coming out of the FLIP simulation and to create multiple layers of secondary particle simulations, are taken to reduce artefacts and shortcomings of the rest of the workflow. The sheer number of layers required in creating these effects is largely due to the fact that each one can only capture a restricted level of detail, either from lack of interaction or fidelity in the methods used. Taking these two modifications as examples, we have demonstrated improvements in both of these areas: increasing the range of detail in spray simulations using the droplet interaction model of Chapter 4; and better handling of fine details as well as larger liquid behaviours in the work described in Chapter 5. Whilst these advances allow significant improvements to the artistic workflow by removing the need for artistic intervention, we also have looked at the development of a tool to help cover remaining issues outside of those we cover explicitly here. This involved the development of a method to make artistic intervention easier, such that in other cases artists would be able to make required modifications quickly and without disrupting their workflow.

The work described in Chapter 3 demonstrates findings that suggest that it is further from a production tool than the other projects explored. However, it does provide a significant body of work and in depth look at the use of model reduction in application to fluid simulation and should offer a good basis for future work. This could arise from any of the suggested avenues we describe in the chapter, or given the recent increase in interest in machine learning and other data-based techniques [Sato et al. 2018a], some similar approach. Alternatively, this work may be useful in other applications similar to the simulation compression work of Jeruzalski et al. [2018].

Many previous works that have looked at improving the ability to create liquids for computer graphics focus on real-time or interactive results, displaying very different quality than that we require in VFX. These works commonly consider the coupling of a collection of methods for the components that make up a liquid effect together, often defining specific blending mechanisms or forces to be applied. Whilst this approach can be successful at defining black-box solutions, the requirement for allowing artistic intervention and for developing robust and flexible tools in VFX has instead driven this work to aim for more tightly defined improvements. In this way, the workflow changes imposed upon artists are minimised, but offer explicit improvements where they take effect. In fact, we have found that these kind of developments can also have much wider scope of influence than is immediately obvious, as they are able to be incorporated in a range of different workflows. Taking the particle surfacing tool developed in Chapter 5 for example, whilst motivated by the requirement to create better results from water simulations, this tool can be used to transform any point set into a smooth surface whilst retaining details present in the distribution. Given the near endless scope of particle-based workflows in VFX, this could be used for anything from reconstructing geometry from a point cloud to modelling the procedural growth of the symbiote suit from *Venom* (the latter of which we have already seen in practice).

The industrial context behind this project has been a key motivator to choices made and working closely with the FX R&D team at DNEG has given a great insight into how to create tools that will have positive impact in production VFX. Apart from the projects we have discussed in detail in this last few chapters, there has been other work undertaken to help drive the development of Dynamo and OpenVDB (Points mainly) during this time as well. The most industrially-focused contribution of this work is that of the expression language OpenVDB AX, documented in Chapter 6, which would have unlikely been possible (or recognised as important) outside of this context. However, given the widening scope of tools such as OpenVDB beyond the VFX and animation industries, the use of this tool may prove to be of similar utility elsewhere as well.

Usage logging has allowed us to track the uptake of some of these developments in production. For a large studio such as DNEG it can be difficult to be aware of when and who is using tools available to them without it. Documentation was made available to the artists on the release of these tools so that personal communication with myself (or others in the FX R&D team) was not always necessary to be able to use the tool successfully. However, alongside user testimony from some senior FX artists at DNEG, this usage logging allows us to see that these developments have seen good adoption and preferred use over the alternative methods they continue to have available.

Probably the largest novel contribution of this work to computer graphics is that of Chapter 4. This work demonstrates a novel algorithm for simulation of droplets using a more physically-based approach than previously taken in graphics. The interactions in these phenomena have often been considered of lesser important to larger spray motions but we show that they are a key driving force in the generation of plausible size and velocity distributions. We are also able to use the simulations results at closer scales than if using other methods as they visibly interact with one another and create size variations in the resulting droplet system. These are qualities that would otherwise have to be manually driven by artists using different layers of simulations or other bespoke approaches. This work suggests that taking new physically-based approaches can help to improve the ability to create realistic phenomena, even for historically artistic-driven effects.

Whilst directed at improving workflows for VFX production specifically, many of the findings of this project could be useful in other fields of computer graphics such as games and virtual reality. For example, the theoretical liquid simulation developments in re-simulation, droplet and particle surfacing remain applicable across these other fields and the technological development of OpenVDB AX offers fast manipulation of VDB data wherever it is used (e.g. general applications of volumetric data).

7.2 Future Work

Whilst the results of these projects demonstrate improvements to the workflows used in production VFX, the approaches taken also open avenues of future research, both immediate and slightly further-afield.

7.2.1 Extending This Work

Re-simulation: Chapter 3 describes a method for reducing simulation time based on the iterative artistic workflow. The results show that whilst it can be effective at speeding up subsequent iterations of a simulation, deviation from the earlier stages of simulation are poorly handled. This method may be possible to extend with the help of basis enrichment methods or similar, but it is unlikely to be applicable to high quality liquid simulation in the near future. Instead it could be interesting to consider this approach as a non-standard mechanism of storage of a simulation like the recent work of Jeruzalski et al. [2018].

Droplets and sprays: Chapter 4 introduces a novel droplet simulation model, efficiently recreating real droplet behaviours of coalescence, separation and breakup. The approach taken requires the introduction of a neighbourhood lookup step, whose cost, albeit somewhat mitigated by a collection of simplifications and approximations that we introduce, significantly outweighs the other stages of the simulation. Improvements to this calculation such as porting the algorithm to the GPU could be beneficial in the future. Furthermore, the translation of the physically-based model to a graphics context, retains a similar parametrisation based on physical quantities such as surface tension and liquid density. Whilst these parameters are able to offer artistic control, it may be interesting to explore whether this model could be simplified to have a more artist-friendly interface without losing its physical plausibility. This droplet simulation model is shown to demonstrate plausible behaviours on large scales, creating emergent varying droplet distributions through its small-scale interactions. However, whilst it captures topological changes on a small-scale, i.e. the creation and breakup of droplet ligaments, it is not yet suitable for use in purely microscopic or close-up simulations due to a lack of droplet deformation handling. This could be introduced using something like spherical harmonics to track droplet surface deformation and oscillation [Ashgriz 2011].

Post-processing simulations: In Chapter 5 the integration of a state-of-the-art particle surfacing method into the VFX is discussed, alongside necessary improvements to better handle the varied inputs and user-control required in visual effects production. In particular, this involves the removal of a possibly destructive user parameter and focuses user-control into intuitive and meaningful parameters. However, further improvements could be made to this model e.g. improvements in robustness to outliers in the covariance matrix (and so deformation) calculation; or better transitions between isolated 'droplets' and particles considered to be in the liquid body, e.g. looking to use the number of neighbours to blend between these two states rather than the discrete jump that occurs in the current method.

Efficient data manipulation: In Chapter 6 a new expression language for manipulating point and volume data is introduced. Whilst existing technical requirements on artists have led them to become familiar with scripting and programming languages, the success of visual programming languages (e.g. many node-based systems such as Houdini/Nuke or even more specifically Houdini VOPs) could suggest such an approach may be beneficial in the future. Also further applications of this language, outside of VDB, would be interesting, such as manipulating meshes or 2D textures.

7.2.2 Other Open Problems in High Quality Liquid Recreation

Apart from the areas tackled by projects in this thesis, there remain other areas of improvement in the wider liquid FX pipeline. These include:

Coupling of simulations/discretisations: A common problem for workflows containing different simulations (e.g. FLIP and SPH or our droplet model) or different discretisations is coupling between them. Consider the coupling of 3D particle-based liquid simulations and 2D procedural oceans (§2.5), whilst some works such as Nielsen and Bridson [2011] have looked to tackle this problem, there are scenarios that are not as well provided for, such as very stormy oceans.

Scalability and data manipulation: Whilst improvements to simulation algorithms and hardware have been shown to allow scaling of simulation resolutions to extremely high levels, there remains an issue of working with the resulting data elsewhere in the pipeline. In practice this means that these increased resolutions are not always used and thus simulation resolutions in production have plateaued to amounts that remain useable elsewhere such as in particle surfacing or mesh creation.

Complex behaviours: Further to those addressed in this thesis, there remain issues in the recreation of complex fluid phenomena such as multi-phase fluids and varying scales of interaction. For example, phenomena such as sea foam evolution and subsequent dynamics have yet to be tackled in any meaningful way.

7.3 Conclusion

In this thesis we have described a collection of projects improving the liquid effects pipeline in visual effects production. We proposed that improving the level of detail captured by physically-based components of these effects would enable more efficient artist workflows than those used currently in production. Previous works have have looked to tackle problems in this area through computational advances, developing faster algorithms or increased accuracy but have often lost sight of the key purpose of the creation of these simulations in the first place, to recreate these physical phenomena for *creative purposes*. Either through lack of focus on the importance of robustness and flexibility, or by tackling problems that are far removed from the realities of day-to-day production, this has led to key problem areas, such as those of recreating splashing liquids, being long overlooked. Whilst tooling may have been missing, the results have been in demand, and so artists have been left to fend for themselves, relying on almost wholly artistically-driven workflows for these complex physical processes. With an invaluable insight into the production workflow through partnership with DNEG, and the freedom to pursue new solutions, this project has been able to tackle these real problems in a working production setting.

The problem of high quality splashing liquids has been shown to be particularly difficult one throughout the VFX and animation industries. Through the partnership with DNEG we have seen the effects of this in reality and gave a demonstration of this with a breakdown of the work required on a single shot from a recent feature film production. In this, we saw that the artist was required to modify and adapt almost every stage of their simulation, as well as layering up to 78 simulated and non-simulated elements to create a plausible final splash effect. This highlighted a collection of key areas in the workflow that could be improved, these were: efficiency, level-of-detail, range of phenomena and art-directability. The subsequent projects we discussed have each tackled issues in at least one of these areas.

First we looked at re-using computationally expensive simulation data to speed up iteration, using model reduction. Focusing only on the most expensive part of a liquid simulation, the pressure projection, we developed 2 methods to allow faster recreation of grid-based fluid simulations, demonstrating significant speed increases in recreating simulations using a reduced collection of fluid velocity fields. However, these were found to be unable to robustly handle modification to the simulation conditions which meant they would be unsuitable as a iteration tool. From this, we tested using these model reduction approaches as a pre-conditioner before collating a collection of possible basis enrichment approaches to be explored in the future. However, following the change of industrial partner and given a better understanding of the issues facing high-end production VFX it became clear that the most important areas of investigation lay in improving other stages in the liquid effects workflow. As data-driven approaches are an increasingly large area of research interest, we believe this will help to inform future work in this domain.

Then, by exploring methodologies beyond the usual approaches used in computer graphics, a novel approach to the troublesome phenomena of liquid droplets was developed. These phenomena have shown to be difficult for researchers in graphics and artists alike to decide on a single reliable workflow. As such, in practical VFX a huge amount of artistic tweaking, modification and experimentation is required to create high quality results. Further still, these workflows can result in artists using layers of simulations to create required levels-of-detail. By basing the behaviour of our model on physically-based and experimentally verified models, we efficiently capture interactions on the scale of a single droplet whilst avoiding requiring any increase in simulation resolution beyond the usual particle-per-droplet. This method demonstrates increased levels-of-detail compared to existing approaches and, driven by physical motivation rather than explicit definition of behaviour by an artist, exhibits large and small scale phenomena of both individual droplets and sprays as a whole. A paper describing this work was presented at SCA 2017.

Whilst choosing the correct method to define the behaviour of these particle simulations is important, following any such simulation high-quality liquid surfaces are still required for final rendering of these results. We then described the work required to implement and integrate a state-of-the-art particle surfacing algorithm into the production pipeline. In particular this required specific decisions and improvements to be made such that the resulting tool would be robust to user inputs and give expected high-quality results at all times. In this case, introducing better handling of geometric properties to remove a potentially troublesome and unintuitive parameter that would vary dependent on the input data. In doing this, we have helped to reduce the requirement on post-processing of liquid surfaces created from particles to create high-quality results whilst fitting into pre-existing workflows. Through tracking the use of this in production we have seen that even though the method has decreased performance compared to some simpler alternatives, the gains in the results are compelling enough to have a positive impact and have led to artists choosing to use this tool because it reduces the overall time taken to reach production quality results. From this, the resulting tool has been heavily used on many recent productions at DNEG such as Venom, The Kid Who Would Be King and Godzilla: King of the Monsters.

Finally, to allow greater artistic control - currently the main driving force behind the creation of production quality liquid effects, we document the development of a new expression language for point and volume data. This gives artists fine control over these simulation elements and facilitates the workflows used widely in production for these, and many other, effects. This has been exposed in various stages of production throughout DNEG and has shown to be of great benefit, even in areas beyond its original focus. It has been made open-source such that it can benefit as many people as possible, and hopefully continue to develop and expand its functionality. Many of the other elements of this project have made use of this tool, contributing greatly towards its development. This has also been used elsewhere throughout the pipeline at DNEG to help speed up developer and artist workflows in various applications, including use on the Academy-Award winning production Blade Runner 2049. A paper describing this work was presented at DigiPro 2018.

On a more general level, this work demonstrates that whilst it is important for industry to be aware of the latest research such that it may benefit from any advances therein, it is equally important that the research being undertaken is aware of how technology is used in practice, to foster novel ideas, approaches and technologies that can be as effective as possible.

The successes of the projects undertaken suggest that our hypothesis was correct. In particular, we have seen artists keen to use the method developed for increased physical plausibility in their droplet and spray simulations, and widespread adoption of the method to better capture fine details arising from their physically-based liquid simulations. Building on this work, we hope to inspire future research to use physical motivations when exploring other liquid (and similar) phenomena such that they may alleviate requirements on artists. In this way, we look forward to seeing a future where no artist has to send off 78 elements for a single effect ever again. This work enforces our belief that the use of physically-based methods will continue to push the boundaries of quality in the effects that make it onto the big-screen.

References

B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. In *ACM Transactions on Graphics (TOG)*, volume 26, page 48. ACM, 2007. URL `https://dl.acm.org/citation.cfm?id=1276437`.

J. D. Anderson and J. Wendt. *Computational Fluid Dynamics*. Springer, 1995.

R. Ando, N. Thurey, and R. Tsuruno. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1202–1214, 2012. URL `https://ieeexplore.ieee.org/document/6171182`.

R. Ando, N. Thürey, and C. Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (TOG)*, 32(4):103, 2013. URL `https://dl.acm.org/citation.cfm?id=2461982`.

R. Ando, N. Thürey, and C. Wojtan. A dimension-reduced pressure solver for liquid simulations. volume 34. EUROGRAPHICS, 2015.

ANSYS. Fluent, 2018. URL `https://www.ansys.com/`.

N. Ashgriz. *Handbook of atomization and sprays: theory and applications*. Springer Science & Business Media, 2011.

N. Ashgriz and J. Poo. Coalescence and separation in binary collisions of liquid drops. *Journal of Fluid Mechanics*, 221:183–204, 1990. URL `https://doi.org/10.1017/S0022112090003536`.

Autodesk. Maya, 2018. URL `https://www.autodesk.co.uk/products/maya/`.

N. Avramoussis, R. Jones, F. Gochez, T. Keeler, and M. Warner. A JIT expression language for fast manipulation of VDB points and volumes. In *Proceedings of the 8th Annual Digital Production Symposium*, page 1. ACM, 2018.

D. Bailey, I. Masters, and M. Warner. GPU fluids in production: A compiler approach to parallelism. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, 2011.

D. Bailey, M. Warner, and H. Biddle. Packing the water pipe. In *ACM SIGGRAPH 2014 Talks*, page 10. ACM, 2014.

D. Bailey, H. Biddle, N. Avramoussis, and M. Warner. Distributing liquids using OpenVDB. In *ACM SIGGRAPH 2015 Talks*, page 44. ACM, 2015.

A. Barber, D. Cosker, O. James, T. Waine, and R. Patel. Camera tracking in visual effects an industry perspective of structure from motion. In *Proceedings of the 2016 Symposium on Digital Production*, pages 45–54. ACM, 2016.

J. Barbič and D. L. James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 982–990. ACM, 2005.

C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. In *ACM Transactions on Graphics (TOG)*, volume 26, page 100. ACM, 2007.

J. Bender and D. Koschier. Divergence-free SPH for incompressible and viscous fluids. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1193–1206, 2017. URL `https://ieeexplore.ieee.org/document/7487018`.

J. Bender, D. Koschier, T. Kugelstadt, and M. Weiler. A micropolar material model for turbulent SPH fluids. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 4. ACM, 2017.

G. Berkooz, P. Holmes, and J. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Rev. Fluid Mech*, pages 539–575, 1993. URL `https://doi.org/10.1146/annurev.fl.25.010193.002543`.

G. L. Bernstein, C. Shah, C. Lemire, Z. Devito, M. Fisher, P. Levis, and P. Hanrahan. Ebb: A DSL for physical simulation on CPUs and GPUs. *ACM Transactions on Graphics (TOG)*, 35(2):21, 2016. URL `https://dl.acm.org/citation.cfm?id=2892632`.

Blender. Blender, 2018. URL `https://www.blender.org/`.

J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982. URL `https://dl.acm.org/citation.cfm?id=357310`.

D. Blythe. The direct3d 10 system. *ACM Transactions on Graphics (TOG)*, 25(3):724–734, 2006.

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. URL `http://www.sciencedirect.com/science/article/pii/0010465588900203`.

P. Brazier-Smith, S. Jennings, and J. Latham. The interaction of falling water drops: coalescence. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 326, pages 393–408. The Royal Society, 1972.

R. Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

R. Bridson, J. Houriham, and M. Nordenstam. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics (TOG)*, 26(3):46, 2007. URL `https://dl.acm.org/citation.cfm?id=1276435`.

J. Budsberg, M. Losure, K. Museth, and M. Baer. Liquids in the croods. 2013.

N. Chentanez and M. Müller. Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Transactions on Graphics (TOG)*, 30(4):82, 2011. URL `https://dl.acm.org/citation.cfm?id=1964977`.

N. Chentanez, M. Müller, and T.-Y. Kim. Coupling 3d Eulerian, heightfield and particle methods for interactive simulation of large scale liquid phenomena. *IEEE Transactions on Visualization and Computer Graphics*, 21(10):1116–1128, 2015a. URL `https://ieeexplore.ieee.org/document/7132780`.

N. Chentanez, M. Müller, M. Macklin, and T.-Y. Kim. Fast grid-free surface tracking. *ACM Transactions on Graphics (TOG)*, 34(4):148, 2015b. URL `https://dl.acm.org/citation.cfm?id=2766991`.

A. J. Chorin. Numerical solution of the navier-stokes equations. *Mathematics of Computation*, 22(104):745–762, 1968. ISSN 0025-5718, 1088-6842. doi: 10.1090/S0025-5718-1968-0242392-2. URL `http://www.ams.org/mcom/1968-22-104/S0025-5718-1968-0242392-2/`.

J. Cornish. The Kid Who Would Be King, 2019. URL `https://www.imdb.com/title/tt6811018/`. Production by Working Title Films.

M. Cuesta. American Assassin, 2017. URL `https://www.imdb.com/title/tt1961175/`. Production by Lionsgate.

F. Da, D. Hahn, C. Batty, C. Wojtan, and E. Grinspun. Surface-only liquids. *ACM Transactions on Graphics (TOG)*, 35(4):78, 2016. URL `https://dl.acm.org/citation.cfm?id=2925899`.

T. De Witt, C. Lessig, and E. Fiume. Fluid simulation using laplacian eigenfunctions. *ACM Transactions on Graphics*, 31(1):1–11, Jan. 2012. ISSN 07300301. doi: 10.1145/2077341.2077351. URL `http://dl.acm.org/citation.cfm?doid=2077341.2077351`.

S. S. DeKnight. Pacific Rim: Uprising, 2018. URL `https://www.imdb.com/title/tt2557478/`. Production by Clear Angle Studios.

Z. DeVito, N. Joubert, F. Palacios, S. Oakley, M. Medina, M. Barrientos, E. Elsen, F. Ham, A. Aiken, K. Duraisamy, et al. Liszt: a domain specific language for building portable mesh-based pde solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 9. ACM, 2011.

M. Dougherty. Godzilla: King of the Monsters, 2019. URL `https://www.imdb.com/title/tt3741700/`. Production by Legendary Entertainment.

Dreamworks Animation. OpenVDB, 2018. URL `http://openvdb.org/`.

D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 736–744. ACM, 2002. URL `https://dl.acm.org/citation.cfm?id=566645`.

D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computers & structures*, 83(6-7):479–490, 2005. URL `https://doi.org/10.1016/j.compstruc.2004.04.024`.

J.-P. Estrade, H. Carentz, G. Lavergne, and Y. Biscos. Experimental investigation of dynamic binary collision of ethanol droplets–a model for droplet coalescence and bouncing. *International Journal of Heat and Fluid Flow*, 20(5):486–491, 1999. URL `https://doi.org/10.1016/S0142-727X(99)00036-3`.

C. Farhat and D. Amsallem. Recent advances in reduced-order modeling and application to nonlinear computational aeroelasticity. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, page 562, 2008.

R. Fattal and D. Lischinski. Target-driven smoke animation. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 441–448. ACM, 2004. URL `http://dl.acm.org/citation.cfm?id=1015743`.

R. Fleischer. Venom, 2018. URL `https://www.imdb.com/title/tt1270797/`. Production by Columbia Pictures Corporation.

N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30. ACM, 2001. URL `http://dl.acm.org/citation.cfm?id=383261`.

N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical models and image processing*, 58(5):471–483, 1996. URL `http://www.sciencedirect.com/science/article/pii/S1077316996900398`.

Foundry. Katana, 2018. URL `https://www.foundry.com/products/katana`.

E. Froemling, T. Goktekin, and D. Peachey. Simulating whitewater rapids in Ratatouille. In *ACM SIGGRAPH 2007 sketches*, page 68. ACM, 2007.

A. Garland. Ex Machina, 2014. URL `https://www.imdb.com/title/tt0470752/`. Production by Universal Pictures.

T. Georjon and R. Reitz. A drop-shattering collision model for multidimensional spray computations. *Atomization and Sprays*, 9(3), 1999. URL `http://www.dl.begellhouse.com/journals/6a7c7e10642258cc,406b807619bd91af,40525640123d40d2.html`.

D. Gerszewski and A. W. Bargteil. Physics-based animation of large-scale splashing liquids. *ACM Transactions on Graphics (TOG)*, 32(6):185, 2013. URL `https://dl.acm.org/citation.cfm?id=2508430`.

D. Gerszewski, L. Kavan, P.-P. Sloan, and A. W. Bargteil. Enhancements to model-reduced fluid simulation. In *Proceedings of the Motion on Games*, pages 201–206. ACM, 2013. URL `http://dl.acm.org/citation.cfm?id=2522634`.

F. Gibou, R. P. Fedkiw, L.-T. Cheng, and M. Kang. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 176(1): 205–227, 2002. URL `https://doi.org/10.1006/jcph.2001.6977`.

R. Hankins, N. Rasmussen, A. Johnson, S. Bowline, and B. Criswell. Dataflow: ILM's framework for procedural geometry generation, simulation authoring, crowds, and more. In *Proceedings of the 2015 Symposium on Digital Production*, pages 7–7. ACM, 2015.

P. Hanrahan and J. Lawson. A language for shading and lighting calculations. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 289–298. ACM, 1990.

F. Harlow and E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12), 1965. URL `https://doi.org/10.1063/1.1761178`.

A. Hay, I. Akhtar, and J. Borggaard. On the sensitivity analysis of angle-of-attack in a model reduction setting. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 1473, 2010.

J.-m. Hong and C.-h. Kim. Controlling fluid animation with geometric potential. *Computer Animation and Virtual Worlds*, 15(3-4):147–157, 2004. URL `https://doi.org/10.1002/cav.17`.

R. Hopper and K. Wolter. The water effects of Pirates of the Caribbean: Dead Men Tell no Tales. In *ACM SIGGRAPH 2017 Talks*, page 31. ACM, 2017.

R. Howard. In the Heart of the Sea, 2015. URL `https://www.imdb.com/title/tt1390411/`. Production by Dune Entertainment.

M. Ihmsen. SPH fluids in computer graphics. 2014.

M. Ihmsen, N. Akinci, G. Akinci, and M. Teschner. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer*, 28(6-8):669–677, 2012. URL `http://link.springer.com/article/10.1007/s00371-012-0697-9`.

M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):426–435, 2014a. URL `https://ieeexplore.ieee.org/document/6570475`.

M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner. SPH fluids in computer graphics. 2014b. URL `http://dx.doi.org/10.2312/egst.20141034`.

Isotropix. Clarisse ifx, 2018. URL `https://www.isotropix.com/`.

J. Iversen and R. Sakaguchi. Growing up with fluid simulation on The Day After Tomorrow. In *ACM SIGGRAPH 2004 Sketches*, page 142, 2004.

T. Jeruzalski, J. Kanji, A. Jacobson, and D. I. Levin. Collision-aware and online compression of rigid body simulations via integrated error minimization. In *Computer Graphics Forum*, volume 37, pages 11–20. Wiley Online Library, 2018.

C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. 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`.

R. Jiang. *Pressure preconditioning using proper orthogonal decomposition*. PhD thesis, Stanford University, 2014.

M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *ACM Siggraph Computer Graphics*, volume 24, pages 49–57. ACM, 1990.

B. Kim, Y. Liu, I. Llamas, X. Jiao, and J. Rossignac. Simulation of bubbles in foam with the volume control method. *ACM Transactions on Graphics (TOG)*, 26(3):98, 2007. URL `https://dl.acm.org/citation.cfm?id=1276500`.

J. Kim, D. Cha, B. Chang, B. Koo, and I. Ihm. Practical animation of turbulent splashing water. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 335–344. Eurographics Association, 2006.

S. Kim, D. J. Lee, and C. S. Lee. Modeling of binary droplet collisions for application to inter-impingement sprays. *International Journal of Multiphase Flow*, 35(6):533–549, 2009. URL `https://doi.org/10.1016/j.ijmultiphaseflow.2009.02.010`.

T. Kim and J. Delaney. Subspace fluid re-simulation. *ACM Transactions on Graphics (TOG)*, 32 (4):62, 2013. URL `http://dl.acm.org/citation.cfm?id=2461987`.

T. Kim, J. Tessendorf, and N. Thürey. Closest point turbulence for liquid surfaces. *ACM Transactions on Graphics (TOG)*, 32(2):15, 2013. URL `https://dl.acm.org/citation.cfm?id=2451241`.

F. Kjolstad, S. Kamil, J. Ragan-Kelley, D. I. Levin, S. Sueda, D. Chen, E. Vouga, D. M. Kaufman, G. Kanwar, W. Matusik, et al. Simit: A language for physical simulation. *ACM Transactions on Graphics (TOG)*, 35(2):20, 2016. URL `https://dl.acm.org/citation.cfm?id=2866569`.

G. H. Ko and H. S. Ryou. Modeling of droplet collision-induced breakup process. *International Journal of Multiphase Flow*, 31(6):723–738, 2005. URL `https://www.sciencedirect.com/science/article/pii/S0301932205000339`.

P. Krysl, S. Lall, and J. E. Marsden. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *International Journal for Numerical Methods in Engineering*, 51(4):479–504, 2001. URL `https://doi.org/10.1002/nme.167`.

M. Kurtz and G. Duda. Foamy creatures: Digital Domain wrangles whitewater for Lord of the Rings. In *ACM SIGGRAPH 2002 conference abstracts and applications*, pages 186–186. ACM, 2002.

J. Lait. Inside Houdini's distributed solver system. In *ACM SIGGRAPH 2016 Talks*, page 42. ACM, 2016.

A. Lee. Life of Pi, 2012. URL `https://www.imdb.com/title/tt0454876/`. Production by Dune Entertainment.

M. Lentine, W. Zheng, and R. Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Transactions on Graphics (TOG)*, 29(4):114, 2010. URL `https://dl.acm.org/citation.cfm?id=1778851`.

B. Liu, G. Mason, J. Hodgson, Y. Tong, and M. Desbrun. Model-reduced variational fluid simulation. *ACM Transactions on Graphics (TOG)*, 34(6):244, 2015. URL `https://dl.acm.org/citation.cfm?id=2818130`.

LLVM Developer Group. LLVM, 2018. URL `https://llvm.org/`.

Los Alamos National Laboratory. KIVA, 2018. URL `https://www.lanl.gov/`.

F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 457–462. ACM, 2004.

F. Losasso, J. Talton, N. Kwatra, and R. Fedkiw. Two-way coupled SPH and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):797–804, 2008. URL `https://ieeexplore.ieee.org/document/4459322`.

J. L. Lumley. *Stochastic tools in turbulence*. Courier Corporation, 2007.

M. Macklin and M. Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4): 104, 2013. URL `https://dl.acm.org/citation.cfm?id=2461984`.

W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: A system for programming graphics hardware in a C-like language. *ACM Transactions on Graphics (TOG)*, 22(3):896–907, 2003. URL `https://dl.acm.org/citation.cfm?id=882362`.

A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74. Eurographics Association, 2010.

A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the adjoint method. In *ACM Transactions On Graphics (TOG)*, volume 23, pages 449–456. ACM, 2004. URL `http://dl.acm.org/citation.cfm?id=1015744`.

O. Mercier, C. Beauchemin, N. Thuerey, T. Kim, and D. Nowrouzezahrai. Surface turbulence for particle-based liquid simulations. *ACM Transactions on Graphics (TOG)*, 34(6):202, 2015. URL `https://dl.acm.org/citation.cfm?id=2818115`.

V. Mihalef, D. Metaxas, and M. Sussman. Simulation of two-phase flow with sub-scale droplet and bubble effects. In *Computer Graphics Forum*, volume 28, pages 229–238. Wiley Online Library, 2009. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01362.x`.

J. Molemaker, J. M. Cohen, S. Patel, and J. Noh. Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–18. Eurographics Association, 2008.

J. J. Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110 (2):399–406, 1994. URL `https://doi.org/10.1006/jcph.1994.1034`.

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

A. Munnannur and R. D. Reitz. A new predictive model for fragmenting and non-fragmenting binary droplet collisions. *International Journal of Multiphase Flow*, 33(8):873–896, 2007. URL `https://www.sciencedirect.com/science/article/pii/S0301932207000419`.

K. Museth. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.*, 32(3), July 2013. URL `https://dl.acm.org/citation.cfm?id=2487235`.

K. Museth. A flexible image processing approach to the surfacing of particle-based fluid animation. In *Mathematical Progress in Expressive Image Synthesis I*, pages 81–84. Springer, 2014.

K. Museth, M. Clive, and N. B. Zafar. Blobtacular: surfacing particle system in Pirates of the Caribbean 3. In *SIGGRAPH sketches*, page 20, 2007.

K. Museth, D. Bailey, J. Budsberg, J. Lynch, and A. Pearce. OpenVDB. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, 2015.

Next Limit. Realflow, 2018. URL `http://www.nextlimit.com/realflow/`.

Y. T. Ng, C. Min, and F. Gibou. An efficient fluid–solid coupling algorithm for single-phase flows. *Journal of Computational Physics*, 228(23):8807–8829, 2009. URL `https://doi.org/10.1016/j.jcp.2009.08.032`.

M. B. Nielsen and R. Bridson. Guide shapes for high resolution naturalistic liquid simulation. In *ACM Transactions on Graphics (TOG)*, volume 30, page 83. ACM, 2011. URL `http://dl.acm.org/citation.cfm?id=1964978`.

M. B. Nielsen and O. Østerby. A two-continua approach to Eulerian simulation of water spray. *ACM Transactions on Graphics (TOG)*, 32(4):67, 2013. URL `https://dl.acm.org/citation.cfm?id=2461918`.

C. Nolan. Inception, 2010. URL `https://www.imdb.com/title/tt1375666/`. Production by Warner Bros.

C. Nolan. Interstellar, 2014. URL `https://www.imdb.com/title/tt0816692/`. Production by Paramount Pictures.

C. Nolan. Dunkirk, 2017. URL `https://www.imdb.com/title/tt5013056/`. Production by Warner Bros.

P. Nordin. *Complex chemistry modeling of diesel spray combustion*. Chalmers University of Technology, 2001.

M. Orme. Experiments on droplet collisions, bounce, coalescence and disruption. *Progress in Energy and Combustion Science*, 23(1):65–79, 1997. URL `https://doi.org/10.1016/S0360-1285(97)00005-1`.

P. O'Rourke. Collective drop effects on vaporizing liquid sprays. Technical report, Los Alamos National Lab., NM (USA), 1981.

P. O'Rourke and F. Bracco. Modeling of drop interactions in thick sprays and a comparison with experiments. *Proceedings of the Institution of Mechanical Engineers*, 9:101–106, 1980.

S. Palmer, J. Garcia, S. Drakeley, P. Kelly, and R. Habel. The ocean and water pipeline of Disney's Moana. In *ACM SIGGRAPH 2017 Talks*, page 29. ACM, 2017.

K. Pan, P. Chou, and Y. Tseng. Binary droplet collision at high Weber number. *Physical Review E*, 80(3):036301, 2009. URL `https://doi.org/10.1103/PhysRevE.80.036301`.

S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, et al. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics (TOG)*, 29(4):66, 2010. URL `https://dl.acm.org/citation.cfm?id=1778803`.

S. Patkar, M. Aanjaneya, D. Karpman, and R. Fedkiw. A hybrid Lagrangian-Eulerian formulation for bubble generation and dynamics. In *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 105–114, 2013.

A. Peer, M. Ihmsen, J. Cornelis, and M. Teschner. An implicit viscosity formulation for SPH fluids. *ACM Transactions on Graphics (TOG)*, 34(4):114, 2015.

D. Penney and N. B. Zafar. Rapid: an artist friendly particle system. In *Proceedings of the 2015 Symposium on Digital Production*, pages 15–19. ACM, 2015.

A. Pentland and J. Williams. *Good vibrations: Modal dynamics for graphics and animation*, volume 23. ACM, 1989.

Pixar Animation Studios. Universal scene description, 2018. URL `https://graphics.pixar.com/`.

J. Qian and C. Law. Regimes of coalescence and separation in droplet collision. *Journal of Fluid Mechanics*, 331:59–80, 1997. URL `https://doi.org/10.1017/S0022112096003722`.

L. Rayleigh. On the instability of jets. *Proceedings of the London Mathematical Society*, 1(1): 4–13, 1878. URL https://doi.org/10.1112/plms/s1-10.1.4.

J. Reisch, S. Marshall, M. Wrenninge, T. Göktekin, M. Hall, M. O'Brien, J. Johnston, J. Rempel, and A. Lin. Simulating rivers in The Good Dinosaur. In *ACM SIGGRAPH 2016 Talks*, page 40. ACM, 2016.

J. Ronning and E. Sandberg. Pirates of the Caribbean: Salazar's Revenge, 2017. URL https://www.imdb.com/title/tt1790809/. Production by Walt Disney Pictures.

S. Sato, Y. Dobashi, and T. Nishita. Editing fluid animation using flow interpolation. *ACM Transactions on Graphics (TOG)*, 37(5):173, 2018a. URL https://dl.acm.org/citation.cfm?id=3213771.

T. Sato, C. Wojtan, N. Thuerey, T. Igarashi, and R. Ando. Extended narrow band FLIP for liquid simulations. In *Computer Graphics Forum*, volume 37, pages 169–177. Wiley Online Library, 2018b.

B. Schäling. *The Boost C++ libraries*. Boris Schäling, 2011.

R. Schmit and M. Glauser. Improvements in low dimensional tools for flow-structure interaction problems: using global pod. In *42nd AIAA Aerospace Sciences Meeting and Exhibit*, page 889, 2004.

M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 1.1), 1999.

L. Shi and Y. Yu. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–236. ACM, 2005. URL http://dl.acm.org/citation.cfm?id=1073401.

S.-H. Shin and C.-H. Kim. Target-driven liquids animation with interfacial discontinuities. *Computer Animation and Virtual Worlds*, 18(4-5):447–453, 2007. URL https://doi.org/10.1002/cav.202.

Side Effects. Houdini, 2018. URL https://sidefx.com.

B. Solenthaler and M. Gross. Two-scale particle simulation. In *ACM Transactions on Graphics (TOG)*, volume 30, page 81. ACM, 2011. URL http://dl.acm.org/citation.cfm?id=1964976.

B. Solenthaler and R. Pajarola. Density contrast SPH interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–218. Eurographics Association, 2008.

O.-Y. Song, H. Shin, and H.-S. Ko. Stable but nondissipative water. *ACM Transactions on Graphics (TOG)*, 24(1):81–97, 2005. URL https://dl.acm.org/citation.cfm?id=1037962.

Sony Pictures Imageworks. Alembic, 2018a. URL http://opensource.imageworks.com/.

Sony Pictures Imageworks. Open shading language, 2018b. URL `http://opensource.imageworks.com/`.

P. Sorrentino. Loro, 2018. URL `https://www.imdb.com/title/tt6748466/`. Production by Indigo Film.

J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999. URL `http://dl.acm.org/citation.cfm?id=311548`.

M. Stanton, Y. Sheng, M. Wicke, F. Perazzi, A. Yuen, S. Narasimhan, and A. Treuille. Non-polynomial galerkin projection on deforming meshes. *ACM Transactions on Graphics (TOG)*, 32(4):86, 2013. URL `https://dl.acm.org/citation.cfm?id=2462006`.

K. Szewc, K. Walczewska-Szewc, and M. Olejnik. Is the motion of a single sph particle droplet/solid physically correct? *arXiv preprint arXiv:1602.07902*, 2016. URL `https://arxiv.org/abs/1602.07902`.

T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki. Realistic animation of fluid with splash and foam. 22:391–400, 2003. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00686`.

Y. Teng, M. Meyer, T. DeRose, and T. Kim. Subspace condensation: full space adaptivity for subspace deformations. *ACM Transactions on Graphics (TOG)*, 34(4):76, 2015. URL `https://dl.acm.org/citation.cfm?id=2766904`.

P. Tennison, T. Georjon, P. Farrell, and R. Reitz. An experimental and numerical study of sprays from a common rail injection system for use in an hsdi diesel engine. Technical report, SAE Technical Paper, 1998.

J. D. Thornton. Directable simulation of stylized water splash effects in 3d space. In *ACM SIGGRAPH 2006 Sketches*, page 94. ACM, 2006.

G. Tomar, D. Fuster, S. Zaleski, and S. Popinet. Multiscale simulations of primary atomization. *Computers & Fluids*, 39(10):1864–1874, 2010. URL `https://doi.org/10.1016/j.compfluid.2010.06.018`.

A. Treuille, A. McNamara, Z. Popović, and J. Stam. Keyframe control of smoke simulations. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 716–723. ACM, 2003. URL `http://dl.acm.org/citation.cfm?id=882337`.

A. Treuille, A. Lewis, and Z. Popović. Model reduction for real-time fluids. *ACM Transactions on Graphics (TOG)*, 25(3):826–834, 2006. URL `https://dl.acm.org/citation.cfm?doid=1179352.1141962`.

S. Trojansky. Raging waters: the rivergod of Narnia. In *ACM SIGGRAPH 2008 talks*, page 74. ACM, 2008.

K. Um, X. Hu, and N. Thuerey. Perceptual evaluation of liquid simulation methods. *ACM Transactions on Graphics (TOG)*, 36(4):143, 2017. URL `https://dl.acm.org/citation.cfm?id=3073633`.

K. Um, X. Hu, and N. Thuerey. Liquid splash modeling with neural networks. In *Computer Graphics Forum*, volume 37, pages 171–182. Wiley Online Library, 2018.

F. Vetrano, C. Le Garrec, G. D. Mortchelewicz, and R. Ohayon. Assessment of strategies for interpolating POD based reduced order models and application to aeroelasticity. *Journal of Aeroelasticity and Structural Dynamics*, 2(2), 2012.

D. Villeneuve. Blade Runner 2049, 2017. URL `https://www.imdb.com/title/tt1856101/`. Production by Columbia Pictures Corporation.

Visual Effects Society. The state of the global visual effects industry. 2013.

E. Vouga, B. Smith, D. M. Kaufman, R. Tamstorf, and E. Grinspun. All's well that ends well: guaranteed resolution of simultaneous rigid body impact. *ACM Transactions on Graphics (TOG)*, 36(4):151, 2017. URL `https://dl.acm.org/citation.cfm?id=3073689`.

H. Wang, P. Mucha, and G. Turk. Water drops on surfaces. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 921–929, 2005.

D. Weber, J. Mueller-Roemer, A. Stork, and D. Fellner. A cut-cell geometric multigrid Poisson solver for fluid simulation. In *Computer Graphics Forum*, volume 34, pages 481–491. Wiley Online Library, 2015.

M. Wicke, M. Stanton, and A. Treuille. Modular bases for fluid dynamics. In *ACM Transactions on Graphics (TOG)*, volume 28, page 39. ACM, 2009.

R. Winchenbach, H. Hochstetter, and A. Kolb. Constrained neighbor lists for sph-based fluid simulations. In *Symposium on Computer Animation*, pages 49–56, 2016.

R. Winchenbach, H. Hochstetter, and A. Kolb. Infinite continuous adaptivity for incompressible SPH. *ACM Transactions on Graphics (TOG)*, 36(4):102, 2017. URL `https://dl.acm.org/citation.cfm?doid=3072959.3073713`.

C. Wojtan, N. Thürey, M. Gross, and G. Turk. Deforming meshes that split and merge. In *ACM Transactions on Graphics (TOG)*, volume 28, page 76. ACM, 2009.

L. Yang, S. Li, A. Hao, and H. Qin. Hybrid particle-grid modeling for multi-scale droplet/spray simulation. In *Computer Graphics Forum*, volume 33, pages 199–208. Wiley Online Library, 2014. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12488`.

T. Yang, J. Chang, B. Ren, M. C. Lin, J. J. Zhang, and S.-M. Hu. Fast multiple-fluid simulation using helmholtz free energy. *ACM Transactions on Graphics (TOG)*, 34(6):201, 2015. URL `https://dl.acm.org/citation.cfm?id=2818117`.

J. Yu and G. Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Transactions on Graphics (TOG)*, 32(1):5, 2013. URL `https://dl.acm.org/citation.cfm?id=2421641`.

Y. Zhu and R. Bridson. Animating sand as a fluid. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 965–972. ACM, 2005. URL `http://dl.acm.org/citation.cfm?id=1073298`.

Appendix: Collision Threshold Derivation

In Chapter 4, we use the collision thresholds originally derived and validated by Ashgriz and Poo [1990]. These were developed to offer physically-accurate, energy-based prediction of the outcome type of a droplet collision based on the colliding state of the pair of droplets. We consider the collision between the pair of spherical droplets i, j with positions $\mathbf{x}_i, \mathbf{x}_j$ velocities $\mathbf{u}_i, \mathbf{u}_j$ and radii $r_i > r_j$.

Stretching Separation

Stretching separation is considered to arise for collisions in which the stretching kinetic energy in the collision is greater than the surface energy of the ligament that forms between the colliding droplets.

Assuming that the collision is such that only a portion of the droplet masses come into contact, we define the volume of interaction of droplet i as $\phi_i V_i$, for V_i the volume of the sphere of radius r_i, centre $\mathbf{x}_i$, cut by the planes parallel to $\mathbf{u}_{ij}$ and tangential to edge of the other droplet sphere, j (Figure 4.5), and similarly for droplet j.

In Ashgriz and Poo [1990], the following equations for ϕ_i, ϕ_j are given:

$$\phi_i = \begin{cases} 1 - \frac{(2-\tau)^2(1+\tau)}{4}, & \text{if } h > r_i. \\ \frac{\tau^2(3-\tau)}{4}, & \text{otherwise.} \end{cases} \tag{1}$$

$$\phi_j = \begin{cases} 1 - \frac{(2\delta-\tau)^2(\delta+\tau)}{4\delta^3}, & \text{if } h > r_j. \\ \frac{\tau^2(3\delta-\tau)}{4\delta^3}, & \text{otherwise.} \end{cases} \tag{2}$$

where $h = (1-X)(r_1 + r_2)$ and $\tau = (1-X)(1+\delta)$. In the main paper, we note that these equations are not valid for the case of a fully overlapped smaller droplet $h > 2r_j$ (where we should have $\phi_j = 1$) and instead suggest calculating these with geometric equations for segments and caps in these cases.

Now, assuming that the remaining non-interacting portions of the droplets continue along their initial trajectory, we calculate the kinetic stretching energy in the entire collision as:

$$\begin{aligned}
\mathbf{E}_{stretch} &= \text{non-interacting KE} + \text{interacting KE} \\
&= \frac{1}{2}\rho[(V_i - V_{i,I})||\mathbf{U}_i||^2 + (V_j - V_{j,I})||\mathbf{U}_j||^2] + \frac{1}{2}\rho[V_{i,I}(\mathbf{U}_i X)^2 + V_{j,I}(\mathbf{u}_j X)^2] \\
&= \frac{1}{2}\rho[(1-\phi_i)V_i||\mathbf{U}_i||^2 + (1-\phi_j)V_j||\mathbf{U}_j||^2] + \frac{1}{2}\rho X^2[V_{i,I}||\mathbf{U}_i||^2 + V_{j,I}||\mathbf{U}_j||^2] \\
&= \frac{1}{2}\rho||\mathbf{U}_{ij}||^2 V_i\left(\frac{\delta^3}{(1+\delta^3)^2}\right)[(1+\delta^3) - (1-X^2)(\phi_j + \delta^3\phi_i)]
\end{aligned} \tag{3}$$

Note that the above equation, as described in Ashgriz and Poo [1990], velocity is formulated in mass-centre coordinates (corrected by Ko and Ryou [2005]), so uses:

$$\mathbf{U}_i = \frac{-\delta^3\mathbf{u}_{ij}}{(1+\delta^3)} \tag{4}$$

$$\mathbf{U}_j = \frac{\mathbf{u}_{ij}}{(1+\delta^3)} \tag{5}$$

but that any use of the relative velocity $\mathbf{U}_{ij}$ remains equal to the usual form, $\mathbf{U}_{ij} = \frac{(1+\delta^3)\mathbf{u}_{ij}}{1+\delta^3} = \mathbf{u}_{ij}$.

The surface energy that opposes this stretching is that of the nominal ligament created from the interacting volume, given by the surface energy associated with a cylinder of height h and volume $V_{interact} = V_{interact,i} + V_{interact,j}$:

$$\begin{aligned}
\mathbf{E}_{surface} &= 2\sigma[\pi h V_{interact}]^{\frac{1}{2}} \\
&= 2\sigma[\pi h(V_{interact,i} + V_{interact,j})]^{\frac{1}{2}} \\
&= 2\sigma[\pi V_i r_i \tau(\phi_i + \delta^3\phi_j)]^{\frac{1}{2}}
\end{aligned} \tag{6}$$

Then if $\mathbf{E}_{stretch} > \mathbf{E}_{surface}$ the collision results in stretching separation.

Considering the equality of the above equation and then rearranging allows definition of a threshold on We as $\text{We}_{stretch}$:

$$\text{We}_{stretch} = \frac{4(1+\delta^3)^2[3(1+\delta)(1-X)(\delta^3\phi_j + \phi_i)]^{\frac{1}{2}}}{\delta^2[(1+\delta^3) - (1-X^2)(\phi_j + \delta^3\phi_i)]} \tag{7}$$

such that the stretching separation threshold is surpassed if $\text{We} > \text{We}_{stretch}$.

Reflexive Separation

For head-on collisions, we check for reflexive separation in a similar way to that of stretching separation. This outcome is said to arise due to a combination of the incident kinetic energies working in opposing directions, and the internal flows induced due to the difference between the colliding droplet surface energies and the nominal coalesced droplet surface energy.

The kinetic energy term is that of the portions of the droplets which directly oppose each other, given by:

$$\mathbf{E}_{counter} = \frac{1}{2}\rho(V_{i,P}||\mathbf{U}_i||^2 + V_{j,P}||\mathbf{U}_j||^2) \tag{8}$$

where the volume $V_{k,P}$ is the volume of the prolate regions of the incident droplets, defined in terms of X by:

$$V_{i,P} = \frac{4}{3}\pi r_i^3(1-\xi)^2(1-\xi^2)^{\frac{1}{2}} \tag{9}$$

$$V_{j,P} = \frac{4}{3}\pi r_i^3(\delta-\xi)^2(\delta^2-\xi^2)^{\frac{1}{2}} \tag{10}$$

where $\xi = \frac{1}{2}X(1+\delta)$. Then the excess surface energy is given by:

$$\mathbf{E}_{excess} = 4\sigma\pi r_i^2[(1+\delta^2)-(1+\delta)^{\frac{2}{3}}] \tag{11}$$

Finally, the remaining portions of droplets try to stretch the combined mass, which reduces the reflexive energy above and so we also include the following stretching energy term:

$$\mathbf{E}_{stretch} = \frac{1}{2}\rho[(V_i - V_{i,P})||\mathbf{U}_i||^2 + (V_j - V_{j,P})||\mathbf{U}_j||^2] \tag{12}$$

The effective reflexive energy is therefore given by:

$$\mathbf{E}_{reflex} = \mathbf{E}_{counter} + \mathbf{E}_{excess} - \mathbf{E}_{stretch} \tag{13}$$

which can be rearranged to:

$$\mathbf{E}_{reflex} = 4\sigma\pi r_i^2\left[(1+\delta^2)-(1+\delta^3)^{\frac{2}{3}} + \frac{We}{12\delta(1+\delta^3)^2}(\delta^6\eta_i+\eta_j)\right] \tag{14}$$

where

$$\eta_i = 2(1-\xi)^2(1-\xi)^{\frac{1}{2}} - 1 \tag{15}$$

$$\eta_j = 2(\delta-\xi)^2(\delta^2-\xi^2)^{\frac{1}{2}} - \delta^3 \tag{16}$$

Reflexive separation is then said to occur when this energy exceeds 75% of the surface energy of the nominal coalesced mass $\mathbf{E}_{surface,k} = 4\sigma\pi(r_i^3+r_j^3)^{\frac{2}{3}}$, i.e. when $\mathbf{E}_{reflex} > 0.75\mathbf{E}_{surface,k}$.

Using these formulations for $\mathbf{E}_{reflex}$ and $\mathbf{E}_{surface,k}$ taking the threshold of the above inequality and rearranging for We gives:

$$We_{reflex} = \frac{3[7(1+\delta^3)^{\frac{2}{3}} - 4(1+\delta^2)]\delta(1+\delta^3)^2}{(\delta^6\eta_i+\eta_j)} \tag{17}$$

and thus a collision with $We > We_{reflex}$ will exhibit reflexive separation.

List of Acronyms

DCC Digital Content Creation program

CG Computer Generated

VFX Visual Effects

FX Effects

FLIP Fluid Implicit Particle

PIC Particle-In-Cell

MAC Marker-and-Cell

PLS Particle Level Set

APIC Affine Particle-in-Cell

SPH Smoothed Particle Hydrodynamics

PCG Preconditioned Conjugate Gradient

ICPCG Incomplete Cholesky Preconditioned Conjugate Gradient

GFM Ghost Fluid Method

MG Multigrid

PBF Position-Based Fluids

PBD Position-Based Dynamics

POD Proper Orthogonal Decomposition

POM Proper Orthogonal Mode

RBD Rigid Body

AX Attribute Expression

JIT Just-In-Time

IR Intermediate Representation

AST Abstract Syntax Tree

SDF Signed Distance Function

WPCA Weighted Principle Component Analysis

SVD Singular Value Decomposition