Université de Montréal

Tools for Fluid Simulation Control in Computer Graphics

par

Arnaud Schoentgen

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

24 Septembre 2021

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Tools for Fluid Simulation Control in Computer Graphics

présentée par

Arnaud Schoentgen

a été évaluée par un jury composé des personnes suivantes :

Fabian Bastin
(président-rapporteur)

Pierre Poulin
(directeur de recherche)

Philippe Meseure
(codirecteur)

Philip Dutré
(membre du jury)

Chris Wojtan
(membre du jury)

David Tonnesen
(examinateur externe)

Résumé

L'animation basée sur la physique peut générer des systèmes aux comportements complexes et réalistes. Malheureusement, contrôler de tels systèmes est une tâche ardue. Dans le cas de la simulation de fluide, le processus de contrôle est particulièrement complexe. Bien que de nombreuses méthodes et outils ont été mis au point pour simuler et faire le rendu de fluides, trop peu de méthodes offrent un contrôle efficace et intuitif sur une simulation de fluide. Étant donné que le coût associé au contrôle vient souvent s'additionner au coût de la simulation, appliquer un contrôle sur une simulation à plus haute résolution rallonge chaque itération du processus de création. Afin d'accélérer ce processus, l'édition peut se faire sur une simulation basse résolution moins coûteuse. Nous pouvons donc considérer que la création d'un fluide contrôlé peut se diviser en deux phases: une phase de contrôle durant laquelle un artiste modifie le comportement d'une simulation basse résolution, et une phase d'augmentation de détail durant laquelle une version haute résolution de cette simulation est générée. Cette thèse présente deux projets, chacun contribuant à l'état de l'art relié à chacune de ces deux phases.

Dans un premier temps, on introduit un nouveau système de contrôle de liquide représenté par un modèle particulaire. À l'aide de ce système, un artiste peut sélectionner dans une base de données une parcelle de liquide animé précalculée. Cette parcelle peut ensuite être placée dans une simulation afin d'en modifier son comportement. À chaque pas de simulation, notre système utilise la liste de parcelles actives afin de reproduire localement la vision de l'artiste. Une interface graphique intuitive a été développée, inspirée par les logiciels de montage vidéo, et permettant à un utilisateur non expert de simplement éditer une simulation de liquide.

Dans un second temps, une méthode d'augmentation de détail est décrite. Nous proposons d'ajouter une étape supplémentaire de suivi après l'étape de projection du champ de vitesse d'une simulation de fumée eulérienne classique. Durant cette étape, un champ de perturbations de vitesse non-divergent est calculé, résultant en une meilleure correspondance des densités à haute et à basse résolution. L'animation de fumée résultante reproduit fidèlement l'aspect grossier de la simulation d'entrée, tout en étant augmentée à l'aide de détails simulés.

Mots-clés: animation basée sur la physique, simulation de fluide, contrôle de fluide, augmentation de détail

Abstract

Physics-based animation can generate dynamic systems of very complex and realistic behaviors. Unfortunately, controlling them is a daunting task. In particular, fluid simulation brings up particularly difficult problems to the control process. Although many methods and tools have been developed to convincingly simulate and render fluids, too few methods provide efficient and intuitive control over a simulation. Since control often comes with extra computations on top of the simulation cost, art-directing a high-resolution simulation leads to long iterations of the creative process. In order to shorten this process, editing could be performed on a faster, low-resolution model. Therefore, we can consider that the process of generating an art-directed fluid could be split into two stages: a control stage during which an artist modifies the behavior of a low-resolution simulation, and an upresolution stage during which a final high-resolution version of this simulation is driven. This thesis presents two projects, each one improving on the state of the art related to each of these two stages.

First, we introduce a new particle-based liquid control system. Using this system, an artist selects patches of precomputed liquid animations from a database, and places them in a simulation to modify its behavior. At each simulation time step, our system uses these entities to control the simulation in order to reproduce the artist's vision. An intuitive graphical user interface inspired by video editing tools has been developed, allowing a nontechnical user to simply edit a liquid animation.

Second, a tracking solution for smoke upresolution is described. We propose to add an extra tracking step after the projection of a classical Eulerian smoke simulation. During this step, we solve for a divergence-free velocity perturbation field resulting in a better matching of the low-frequency density distribution between the low-resolution guide and the high-resolution simulation. The resulting smoke animation faithfully reproduces the coarse aspect of the low-resolution input, while being enhanced with simulated small-scale details.
Keywords: physics-based animation, fluid simulation, fluid control, upresolution

Contents

List of Figures

Remerciements

Avec la rédaction de cette thèse s'achèveront bientôt plusieurs années de doctorat. Bien qu'ayant été dure à plusieurs niveaux, cette expérience a été grandement allégée par la présence et le support d'un certain nombre de personnes que j'aimerais remercier ici.

Pour commencer, merci infiniment à Pierre Poulin d'avoir su être le directeur de thèse qu'il a été. Merci de m'avoir donné l'opportunité de faire ce premier stage au LIGUM, et de revenir quelques mois plus tard pour débuter une thèse en me donnant une liberté totale quant à mes axes de recherche. Son soutien inconditionnel, son intuition, nos discussions enrichissantes et son enthousiasme pour tout ce qui touche de près ou de loin à l'informatique graphique (et au kimchi) ont été des sources de motivation continuelles et inépuisables. Avoir un tel directeur de thèse était une chance que je souhaite à toute personne voulant poursuivre un doctorat.

Un grand merci à mon second directeur de thèse Philippe Meseure, pour son soutien à tout épreuve. Son aide, sa gentillesse et sa prévenance sont autant d'aspects qui, malgré la distance, ont fait de cette cotutelle une expérience positive. Merci encore Philippe!

Merci à Emmanuelle Darles pour sa présence, en particulier en début de thèse.

Merci aux membres de mon jury d'avoir accepté de participer à l'évaluation des travaux effectués dans le cadre de mon doctorat. Qu'il s'agisse de ma thèse ou de la soutenance.

Merci à Bernhard Thomaszewski et Misha Bessmeltsev d'avoir contribué à faire du LIGUM un endroit si propice à la réalisation d'une thèse en informatique graphique. Nombre de discussions que j'ai pu avoir avec vous ont été inspirantes.

Merci à mon ami et collègue Jonas Zehnder pour le temps de qualité que l'on a passé ensemble. J'ai beaucoup appris en travaillant avec toi, en particulier sur le second projet présenté dans cette thèse. Nos échanges variés dans la cuisine du labo me manqueront.

Merci à Adrien Dubouchet, mon ami de première heure au LIGUM. La vie au labo (et en dehors) n'aurait pas été la même sans toi.

De même, merci à Alexandre Jubert et Melino Conte d'avoir fait du LIGUM un endroit si vivant et si plaisant.

De manière générale, merci à tous les membres du LIGUM que j'ai eu la chance de côtoyer, étudiants et visiteurs, anciens et récents. Merci à Antoine, Bowen, Bruno, Caio, Chaitanya, Charles, Cihan, Cynthia, Dabid, David, Elsa, Étienne, Guochao, Ivan, Jean, Jean-Philippe, Joël, Keith, Kevin, Kirill, Luis, Masha, Mathieu, Nicolas, Olivier, Pengbin, Rémi, Shuhei, Sonia, Takuto, Vincent, William et Yangyang.

Durant mon doctorat, j'ai été chanceux de recevoir le support de personnes extérieures au labo. Merci notamment à mes amis du Canada pour tous les beaux moments passés ensemble, loin d'un ordinateur.

Je ne pouvais pas écrire ces quelques lignes sans remercier tout particulièrement Océane, pour sa présence et son soutien indéfectible. Elle (et plus récemment ma filleule Léane dite "Choubie") a été un soutien de taille à de nombreux moments. Merci sincèrement pour ça.

Un immense merci à Nada pour son écoute, son support et sa patience. Être avec quelqu'un qui termine un doctorat n'est pas simple tous les jours. Merci de m'avoir supporté et d'avoir su enrichir ma vie à de nombreux moments où j'en avais besoin. Je n'aurais pas pu espérer meilleure partenaire pour finaliser cette thèse.

Pour finir, mes remerciements les plus chaleureux vont à ma famille. Merci à mon cousin Etienne et à Mélanie pour leur soutien. Votre visite à Montréal m'avait fait le plus grand bien.

Merci à mes grands-parents Huguette, Simone, Paul et Norbert. Merci pour votre amour et pour tous les beaux moments passés ensemble. J'espère vous rendre fiers.

Merci à Sarah, à Eric (et à Gildas) pour leur présence et leur soutien. Même sur un autre continent et à des milliers de kilomètres, j'ai toujours reçu le support et l'amour de ma soeur et de mon frère. Merci à vous deux.

Enfin et surtout, merci à mes parents Pierre et Sylvie de m'avoir donné des outils essentiels et inestimables, en m'éduquant et en m'inculquant des valeurs que je continue de porter et dont je suis fier. Merci de m'avoir soutenu de toutes les manières possibles et imaginables. Sans vous, rien de tout cela n'aurait été possible. Merci du fond du coeur.

Chapter 1

Introduction

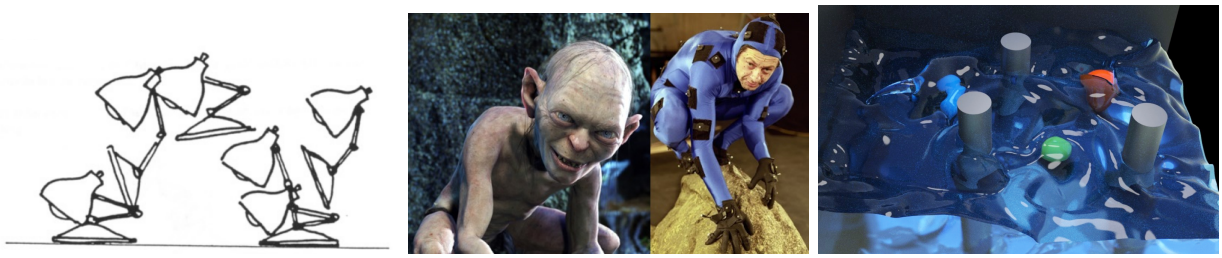
1.1. Context

Humans have always been ingenious when looking for new ways of depicting the world surrounding them. Computer graphics is a particularly fascinating field that fuses many theoretical concepts from several fields for the purpose of creating convincing digital models and images as efficiently and precisely as possible. Such typical fields include among others computer science, mathematics, mechanics, optics, biology, and electromagnetism. The resulting technologies are used in many fields of application, including, but not restricted to, the entertainment industry, medicine, and engineering. Since the production of *A Computer Animated Hand* [**21**], the first computer-animated short film co-produced by Catmull and Parke in 1972, the quality of Computer-Generated Images (CGI) never stopped to get better and better, reaching nowadays an unprecedented level of realism as shown in Figure 1.1.1.

Fig. 1.1.1. Shot from Disney's feature film *Moana* [**25**] (©2016 Disney).

Over time, sub-fields of computer graphics have taken shape, including rendering, geometry processing, digital fabrication, and computer animation. The latter formulates how matter and objects move and deform in a virtual world. Once generated over time, animated objects can be displayed with traditional rendering techniques in order to produce an image sequence showing the animation from a virtual camera point of view. Three main categories of computer animation approaches can be distinguished:

- **Keyframing** generates animations by interpolating user-provided states describing the system at specific moments. Note that forward or inverse kinematics can be used to determine the states of the remaining components of a kinematic chain (i.e., rigid bodies connected by joints) when defining the pose of one component of a system such as a skeleton.
- **Motion capture** uses special equipment to record real-world motion before transferring it to an object.
- **Procedural animation** formulates the motion of a system using a set of rules. In the case where these rules are differential equations describing the dynamics of a real world, we are referring to these techniques as physics-based animation.



(a) Keyframing. (b) Motion capture. (c) Procedural animation.

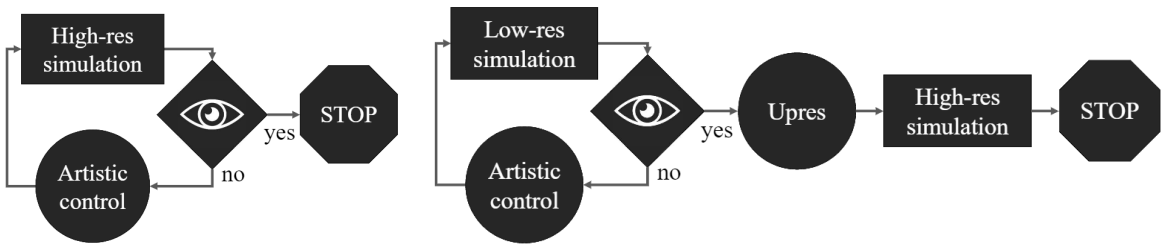
Fig. 1.1.2. Computer animation approaches. (a) Key poses of Pixar's Luxo lamp from Lasseter [**64**] defined at a few times to describe an animation. (b) The digital character of Gollum in *The Lord of the Rings—The Two Towers* [**50**], and the actor whose movements were captured using sensors to animate Gollum. (©2002 Warner Bros. Pictures) (c) Liquid animation from Yang et al. [**136**] generated by simulation.

Each approach, illustrated in Figure 1.1.2, comes with typical use cases and weaknesses. Physics-based animation can especially lead to systems of very complex and realistic behaviors that could not be easily animated using other types of method, for example scenes involving pieces of clothing, ropes, fracturing rigid objects, deformable shapes, hair and fur, liquids, fires, etc. Unfortunately, numerically solving the physics equations of motion for a given type of material is often computationally expensive because fine spatial and temporal

discretizations are usually required to obtain convincing results. Most of the time, it is also challenging to inject art-direction in a simulation. Since the resulting animation depends on its initial state, time integration scheme, and underlying physics, a user could try to modify the initial conditions of a dynamic system in order to obtain a targeted behavior. When dealing with chaotic systems, this approach is too limited and too unpredictable, suggesting the need of specialized tools to control dynamic systems.

Amongst physical models, fluid simulation brings up particularly difficult problems to the control process, mainly because of the complexity and nonlinearity of the supporting Navier-Stokes equations. One consequence of this nonlinearity is the Butterfly effect, a concept expressing the potentially large impact that a small perturbation may have on a nonlinear dynamic system after a given period of time. For this reason, using a handmade force field or modifying initial conditions of a liquid simulation is quite often unpredictable when a user wants to generate a target animation. Many methods and tools have been proposed to convincingly and efficiently simulate and render different types of fluids, but too few methods provide intuitive ways to art-direct them. However, being able to control a fluid in order to make it behave according to an artist's vision is crucial to enhance the creative process. This is particularly true in a production context where artists need robust tools to generate fluid animations that do not need to be entirely physically correct. When animators want to obtain a target animation, they often favor more controllable tools rather than simulation. For instance, in order to create waves in the movie *Surf's Up*, animators have designed a procedural system because they found it more controllable [**20**]. To produce splashes in Pixar's feature film *Ratatouille*, artists have used NURBS to accurately represent the liquid [**18**]. More recently, animators have exploited geometry deformers as well as nonsimulation-based procedural techniques to produce ice cream effects for the movie *Zootopia* [**19**]. These are good high-profile examples that justify why control techniques should be improved in order to better inject art-direction into simulation-based animation.

A fluid control method must be effective to get adopted, meaning that it should generate art-directed fluids that behave in accordance with the artist's vision. At the same time, the resulting animation should exhibit fluid-like features, showing secondary motions like vortex regions for smoke, or ripples and splashes for liquids. Moreover, a control method should be intuitive and practical, so that a wide range of animators (i.e., technical and nontechnical) could easily specify the inputs needed to generate a controlled animation. Last but not least, such a method should be as efficient as possible in order to not hamper the creative process. This is particularly true as usually, artists follow a trial-and-error approach when controlling a simulation, using whatever tool they have access to. At each iteration



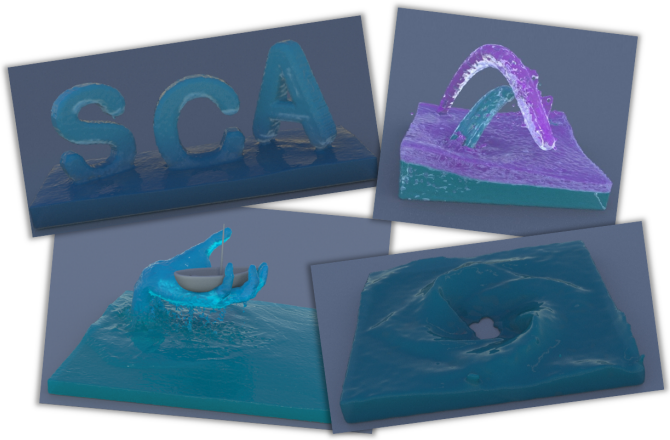
(a) Control at high resolution (b) Control at low resolution, followed by an upresolution step

Fig. 1.1.3. Editing at high resolution versus editing at low resolution before upresolution. (a) Diagram depicting a fluid editing process based on a trial-and-error method using a high-resolution simulation, resulting in long iterations of the fluid art-direction. (b) Same process using a low-resolution model, followed by an upresolution stage. In this case, the simulation is much faster, resulting in shorter iterations of the creative process.

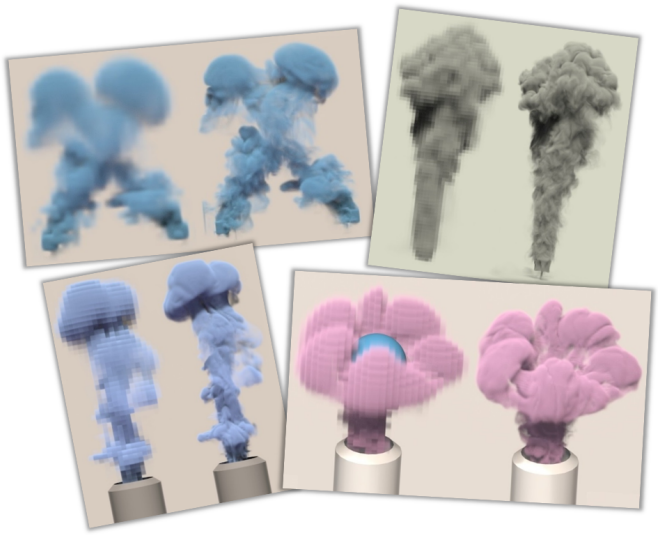
of the process, a user observes how the controlled fluid behaves, and modifies the control inputs in order to converge to a target animation. Each iteration of this creative process requires a full controlled simulation, which can take a long time when using a high-resolution fluid simulation. When dealing with a resolution achieving interactive simulation, a control method should ideally add a small overhead to the simulation, allowing an artist to have a quick visual feedback on the edited simulation. Unfortunately, interactive fluid simulation and, therefore, interactive fluid control is unreachable at high resolutions. A classical solution to tackle this issue is to split the fluid control process in two stages as shown in Figure 1.1.3. During a first control stage, a user can control a low-resolution fluid, iteratively refining the user inputs used to control the simulation and getting quick visual feedback of the resulting animation. During a second upresolution stage, a high-resolution fluid is generated, hopefully following closely enough the low-resolution simulation but enhanced with fine details.

1.2. Content and Organization

This thesis presents two fluid control methods that can be used by an artist to implement such a two-stage control process. The first method introduces a new system that can be used by an artist to intuitively and efficiently control a particle-based liquid simulation using precomputed patches of animated liquids, called templates. The second method presents a new strategy to perform smoke simulation upresolution by accurately tracking smoke density of a low-resolution input. Both methods have been published in journals and presented in associated international conferences [**105, 106**].



(a) Our template-based control of particle-based liquids.



(b) Our upresolution technique for grid-based smoke simulation.

Fig. 1.2.1. Results generated using the two systems developed in this thesis. (a) Particle-based Liquid Control using Animation Templates [**105**]. (b) A Density-Accurate Tracking Solution for Smoke Upresolution [**106**].

This thesis is organized as follows. Chapter 2 introduces some key concepts of fluid simulation as well as an overview of the previous work in the fields of fluid control, editing, synthesis, and upresolution. Chapter 3 and Chapter 4 present respectively each project mentioned above and depicted in Figure 1.2.1, including a discussion about our motivation, related work, development of the method, discussion of the results, and future work. Finally, we draw a conclusion in Chapter 5 and draft future work in the field of fluid control.

Chapter 2

Related Work

Fluid simulation has been an active field of research in computer animation for many years. Almost immediately after its introduction to computer graphics, researchers have started to investigate ways of performing control over such simulations, given the difficulties for animators to reach certain target effects.

The methods and software designed during our doctoral studies have been inspired by many research papers. In this chapter we first cover what we consider to be some key contributions to fluid simulation. Note that we do not aim at providing a comprehensive and complete survey of fluid simulation, but instead we discuss contributions related to the simulation techniques that inspired more closely our research. We then provide a more comprehensive overview of fluid control, editing, interpolation, and upresolution techniques, which are at the core of our work.

2.1. Fluid Simulation

Producing a plausible fluid animation is a challenging task. Traditional keyframe animation has proven to be too tedious to be used for this purpose. Non-simulation-based procedural techniques have also been proposed to generate procedural fluid flows [**16**] or oceans [**44, 121, 84, 53, 52, 51**]. These methods are efficient but cannot produce complex liquid behaviors such as liquid splashes. In order to obtain this type of animations, one should rely on simulation-based techniques.

In computer animation, most fluid flows of interest are governed by the incompressible Navier-Stokes equations, a set of partial differential equations named after the engineer and physicist Claude-Louis Navier and the physicist and mathematician George Gabriel Stokes. These equations are usually written as:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \frac{1}{\rho} \nabla p = \mathbf{g} + \nu \nabla \cdot \nabla \mathbf{u}, \tag{2.1.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2.1.2}$$

with velocity $\mathbf{u}$, density ρ, pressure p, gravity $\mathbf{g}$, and kinematic viscosity ν. ∇, $\nabla \cdot \nabla$, and $\nabla \cdot$ are respectively the gradient, the Laplacian, and the divergence operators. Equation (2.1.1) is called the momentum equation. It describes how a fluid locally accelerates when forces are acting on it. Although real fluids can slightly change their volume, computer graphics usually makes the approximation that liquids and gases are incompressible. This property is expressed by Equation (2.1.2), called the incompressibility equation. Unfortunately and because of their complexity, no exact solution to this pair of equations has been found yet. In order to approximate a solution, these equations are discretized in both space and time. The spatial discretization can be done using either a grid (Eulerian approaches), particles (Lagrangian approaches), or some hybrid strategy. Most of the time, a strategy called splitting is used to approximate a solution to these equations. It consists in splitting up a complex equation into several components, each of them solved using an algorithm specifically designed for it. Usually, a pressure-solve step is used to find pressure values resulting in a velocity field as divergence-free as possible. An advection step is used to advect physical quantities through a velocity field.

2.1.1. Eulerian Simulation

An Eulerian approach considers the evolution of physical quantities describing the state of a fluid at fixed positions in space. The seminal work of Foster and Metaxas [**35**] that introduced fluid simulation to the graphics community uses this strategy. Their method approximates the behavior of a fluid by solving the Navier-Stokes equations on a grid, in a way that would satisfy an animator's needs. When discretizing the evolving physical quantities in space, a staggered Marker-and-Cell (MAC) grid model [**42**] is preferred to a classical collocated grid. Storing the physical quantities at different locations (velocity at the borders of each cell, density and pressure at the center, as shown in Figure 2.1.1) allows for accurate computation of the differential operators using central difference.

Stam [**115**] uses a semi-Lagrangian technique to perform the advection, making the simulation algorithm unconditionally stable. Unfortunately, the use of a low-order spatial interpolation (e.g., linear interpolation) in the semi-Lagrangian advection leads to numerical diffusion, and therefore, to loss of energy and detail. Fedkiw et al. [**31**] tackle the numerical diffusion problem by using a sharper form of spatial interpolation based on a Catmull-Rom interpolation as well as vorticity confinement. Since then, many methods have been developed using advection schemes as detail-preserving as possible [**58, 109, 75, 141, 138, 80**].

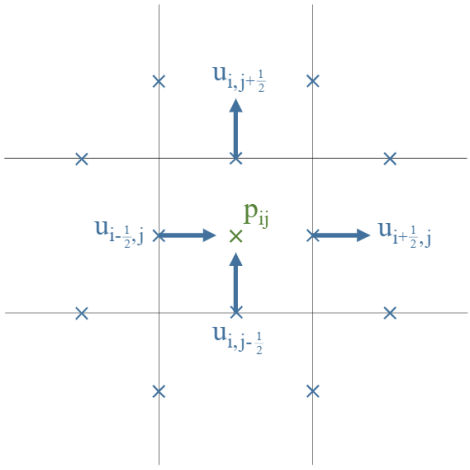


Fig. 2.1.1. A 2D cell from a staggered MAC grid, and locations of physical quantities.

Enright et al. [**29**] generate particles on both sides of a liquid surface, using them to maintain an accurate representation of the surface. In its most popular and simple form [**115**], a grid-based simulation is given by Algorithm 1.

Algorithm 1: Simple grid-based simulation

```
initializeSimulation();
while simulate do
    advectDensity();              // advection of density through velocity
    advectVelocity();             // advection of velocity through velocity
    applyForces();    // apply forces (e.g., buoyancy) to velocity field
    solvePressure();                          // solve pressure equation
```

Note that each time step of a simulation corresponds to advancing the simulation of a short period of time Δt. The smaller this Δt, the closer the simulation to the exact solution, but the longer the entire computation time. A necessary condition on Δt for convergence is given by the Courant-Friedrichs–Lewy (CFL) condition, that imposes an upper limit on the size of the time step Δt. This condition is given by:

$$\Delta t \leq \frac{C\Delta x}{|\mathbf{u}|}, \tag{2.1.3}$$

with C a small constant integer and Δx the size of a grid cell. For a more in-depth review about Eulerian simulations, including details concerning the formulation and solve of the pressure equation, we strongly refer the reader to the excellent book from Bridson [**15**].

2.1.2. Lagrangian Simulation

Instead of considering the evolution of physical quantities at specific positions in space, the Lagrangian approach follows individual fluid parcels as they evolve in space and time. Among Lagrangian methods, the Smooth Particle Hydrodynamics (SPH) method remains the most popular. Initially developed by Monaghan [**76**] for astrophysical problems, SPH has been extended to deal with free-surface incompressible flows by Monaghan [**77**]. After its introduction to the computer graphics community to simulate gaseous phenomena [**116**] and deformable bodies [**27**], Müller et al. [**78**] extend this method to simulate fluids with free surface. The same year, Premože et al. [**94**] propose to solve the Navier-Stokes equations using the Moving-Particle Semi-Implicit method.

In an SPH system, each particle i carries a given amount of a physical quantity A. At a position $\mathbf{x}$ in space, we can estimate the physical quantity $A(\mathbf{x})$ by summing the contribution of every particle, which gives the following expression:

$$A(\mathbf{x}) = \sum_i m_i \frac{A_i}{\rho_i} W(||\mathbf{x} - \mathbf{x}_i||, h), \tag{2.1.4}$$

with $\mathbf{x}_i$, m_i, and ρ_i respectively the position, mass, and density of the i-th particle. W is a kernel function used to monotonically decrease the contribution of a neighboring particle as we move away from it, as shown in Figure 2.1.2. For practical reasons, we use a smoothing

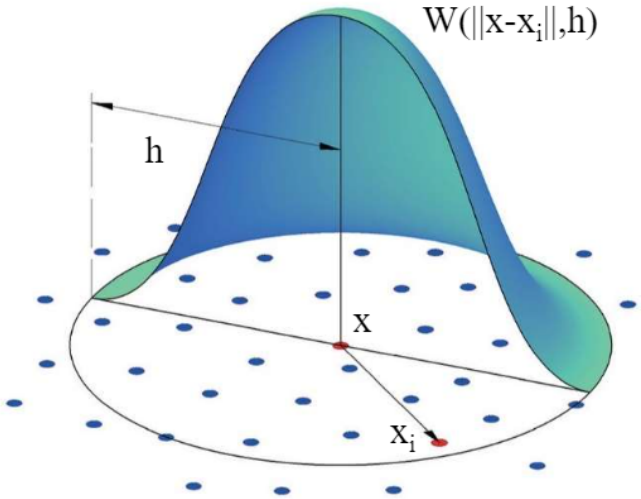


Fig. 2.1.2. A standard smoothing kernel with finite support.

kernel with a finite support radius h. Note that the most widely used 3D kernel can be

found in a paper by Müller et al. [**78**]. Because particles further away than h from position $\mathbf{x}$ are not considered when estimating $A(\mathbf{x})$, it is standard to store all the particles in an acceleration structure (e.g., a hashmap) to allow for quick neighborhood search. In addition to compute physical quantities, the SPH method allows us to compute the gradient and Laplacian of physical quantities. Note that custom kernels have been proposed to perform these computations. At each simulation time step, pressure, viscosity, surface tension, and external forces (e.g., gravity) are computed and used to set the acceleration of each particle, before updating its velocity and position. A simple SPH simulation is given by Algorithm 2.

Algorithm 2: Simple particle-based simulation

```
initializeSimulation();
while simulate do
    updateAccelerationStruct();    // store particles in acceleration structure
    for p ∈ particles do
        computeDensity();                // compute density using equation 2.1.4
        computePressure();    // compute pressure using a stiff equation of state
    for p ∈ particles do
        computePressureForce();                      // compute pressure forces
        computeViscosityForce();                    // compute viscosity forces
        computeSurfaceTensionForce();         // compute surface tension forces
        computeExternalForce();       // compute external forces (e.g., gravity)
    for p ∈ particles do
        updateVelocity();                // perform time-integration on velocity
        updatePosition();                // perform time-integration on position
```

Although performing advection in a Lagrangian formalism is easier than doing so in an Eulerian formalism, the pressure computation is far from trivial and remains one of the bottlenecks of Lagrangian methods. Many solutions have been proposed to estimate pressure forces that minimize volume variations of the simulated fluid. Noniterative equation of state (EOS) methods, such as WCSPH [**9**], propose to compute pressure directly from density. Usually, pressure is expressed in terms of deviation of the estimated density from the rest density in a stiff equation of state. The stiff Tait equation of state is widely used for liquids, for example by Monaghan [**77**]. Iterative variants of EOS solvers are also found in the literature. In this type of method, intermediate velocities and positions are computed using nonpressure forces. Pressures [**114**], pressure forces [**43**], or position changes [**68**] are then iteratively updated while minimizing density errors. Instead of using a state equation, one can solve a pressure Poisson equation (PPE). After advecting particles using nonpressure

forces, pressure values can be computed by solving a discretized PPE in order to correct the variation between each particle density and a rest density [**110**]. Ihmsen et al. [**47**] introduce the Implicit Incompressible SPH (IISPH) method, where a symmetric SPH pressure force is used to obtain a discretized form of a PPE. The pressure field is then iteratively computed using relaxed Jacobi. The Divergence-Free SPH (DFSPH) method, introduced by Bender and Koschier [**11**], maintains not only constant density but also a divergence-free velocity field using two pressure solvers.

Note that extensive research has been carried out to improve many aspects of Lagrangian simulations, such as two-way coupling [**1**], highly viscous liquid simulation [**24, 90, 63**], interactions with porous objects [**65**], or multiple phases and fluid mixing [**98**]. We will not cover these topics but we refer the reader to both the comprehensive state-of-the-art report from Ihmsen et al. [**48**], and the course notes of Koschier et al. [**62**] for further details about SPH methods.

2.1.3. Hybrid Simulation

Both Eulerian and Lagrangian formalisms have their own strengths and weaknesses. Hybrid methods have been proposed to take advantage of the strengths of the grid-based and particle-based representations.

Foster and Fedkiw [**34**] use a grid-based method in addition to inertialess particles to correct the result of the advection step when simulating water. The Particle-in-Cell (PIC) method, proposed by Harlow [**41**] to simulate compressible flow in a computational fluid dynamics (CFD) context, executes the simulation algorithm on a grid, except for the advection step, which is performed using particles. Unfortunately, repeatedly interpolating physical quantities leads to excessive numerical diffusion. The Fluid-Implicit-Particle (FLIP) method, proposed by Brackbill and Ruppel [**14**], tackles this issue by considering the particles as the main representation. Unfortunately, this method suffers from stability issues. Zhu and Bridson [**143**] introduce to computer graphics a method called PIC/FLIP, mixing these two strategies to simulate liquids and sand. The resulting method described by Algorithm 3 uses a weighted proportion of PIC and FLIP, letting the user tune how viscous the simulated liquid should be. Ferstl et al. [**32**] reduce the number of particles used in a FLIP simulation by only using particles within a narrow band of the liquid surface, while representing the remaining liquid volume using a regular grid.

The Affine Particle-In-Cell (APIC) method [**54**] greatly reduces the numerical diffusion of PIC by augmenting each particle with a locally affine description of the velocity. Stomakhin et al. [**117**] introduce the Material Point Method (MPM) to computer graphics in order

Algorithm 3: Simple hybrid simulation

```
initializeSimulation();
while simulate do
    particlesToGrid(U);                    // transfer data from particles to grid
    U^old = U;
    applyForcesGrid(U);                    // apply forces (e.g., gravity) on grid
    solvePressureGrid(U);                  // enforce incompressibility on grid
    U^delta = U - U^old;                   // compute change in velocity
    for p ∈ particles do
        u_PIC = interpolate(U, p);         // PIC: interpolate velocity from grid to
         particles
        u_FLIP = interpolate(U^delta, p);  // FLIP: interpolate change in velocity
         from grid to particles
        updateVelocity(u_PIC, u_FLIP, α);  // compute weighted average of PIC/FLIP
        updatePosition();                  // perform time-integration on position
```

to simulate snow. MPM, originally designed to extend FLIP to solid-mechanics problems requiring compressibility, has been applied to simulate sand and water mixtures [**120**]. More details about hybrid techniques can be found in the book of Bridson [**15**].

2.2. Fluid Simulation Control

Being able to generate a physics-based animation corresponding to a targeted behavior is crucial in the animation and VFX industries. Methods have been proposed to constrain character animations [**129**], rigid bodies [**8, 38, 93, 22, 126**], and deformable bodies [**130, 6, 7, 108, 66**]. However, fluid simulation control presents distinct challenges that have been tackled by many researchers.

Controlling a fluid by modifying the initial conditions of a simulation is hardly an option when a user wants to achieve a targeted animation. At the other end of the spectrum, over-constraining the simulation system (e.g., overriding the velocity field) does not produce plausible secondary motions usually achieved with a physical simulation. Finding a good balance between these two extrema is crucial when designing a control system. This task is complex, mainly due to the nonlinearity of the Navier-Stokes equations. In addition to the effectiveness of a control strategy (i.e., obtaining a target result), its performance is also crucial. The cost of controlling a simulation is added to the simulation cost. Finding an efficient control method is therefore necessary not to hamper the creativity of an artist.

In their original work, Foster and Metaxas [**36**] controlled a fluid by directly modifying its physical properties, such as pressure and surface tension. The resulting system was able to produce simple animations (e.g., a controllable fountain), but could not be used to make the fluid reach complex targets. However, it gave birth to a new, complex, and active field of research in computer graphics. This section covers what we consider to be the main contributions in the field of fluid simulation control.

Different classifications could be proposed to categorize fluid simulation control methods. For instance, one could make a distinction between forward-control methods, that apply a direct control over the velocity field or pressure field, and inverse-control methods, for which a user creates targets to match, letting the system generate forces to achieve the targeted effect. Instead, we decided to classify the methods depending on their control strategy, because we found that methods within a same category had more in common, and therefore, could be more easily comparable. Note that all these methods aim at controlling a fluid during its simulation. In our classification, we make a distinction between the following classes:

- Optimal controllers [**125, 71, 86, 88, 49**]
- Controllers using empirical forces [**37, 45, 30, 3, 89, 111, 112, 61, 139, 135, 119**]
- Controllers using control particles [**95, 124, 69, 140, 67**]
- Controllers using custom boundary conditions [**96, 118**]

- Manipulation of the simulation domain [**103, 87**]

In the following sections, we will describe the approaches and results in each of these categories.

2.2.1. Optimal Controllers

Optimal controllers formulate the fluid-control problem as a numerical optimization over the dimension of possible forces, constrained by the Navier-Stokes equations. A smoke simulation can be described as a succession of states $\mathbf{q}_t = \{\rho_t, \mathbf{u}_t\}$, recursively computed using $\mathbf{q}_t = f(\mathbf{q}_{t-1})$ and the initial condition $\mathbf{q}_0$. The resulting simulation is given by the sequence of states $\mathbf{Q} = [\mathbf{q}_0, \mathbf{q}_1, ..., \mathbf{q}_n]$. A first control strategy consists in finding the minimal amount of control that should be applied to a simulation in order to achieve a targeted behavior. We can design an objective function that estimates how well a simulation meets the goals. In practice, this type of method is used to make smoke reach user-defined "keyframes" $\mathbf{q}_t^* = \{\rho_t^*, \mathbf{u}_t^*\}$ at given times during the simulation. Such an objective function is usually of the form:

$$\phi = k_m \sum |(\mathbf{q}_t - \mathbf{q}_t^*)|^2 + k_s \sum |\mathbf{f}_t|^2 , \tag{2.2.1}$$

where the first term is designed to penalize simulated states too far from the target, while the second term fosters a solution with minimal control forces $\mathbf{f}_t$. Let us assume a control strategy (e.g., parametric forces) from which we extract a vector of control parameters $\mathbf{w}$. We can define the vector function $\mathcal{F}$ as $\mathcal{F}(\mathbf{Q},\mathbf{w}) = \mathbf{Q} = [\mathbf{q}_0, f(\mathbf{q}_0,\mathbf{w}), ..., f(\mathbf{q}_{n-1}, \mathbf{w})]$. We look for the value of $\mathbf{w}$ such that the objective function that we defined is minimal. The resulting control parameters $\mathbf{w}^*$ can be found by solving:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \phi(\mathcal{F}(\mathbf{Q},\mathbf{w}), \mathbf{w}). \tag{2.2.2}$$

If the simulation algorithm is differentiable, one can compute $\nabla\phi$ and minimize function ϕ using a gradient-based optimization method. In practice, a limited-memory quasi-Newton optimization technique can be used. Compared to Newton's method, this type of technique approximates the second derivative (i.e., Hessian matrix) of the objective function by using a small set of recent gradient vectors stored in memory in order to speed up convergence. Therefore, we only have to compute the gradient of the objective function at each iteration of the numerical optimization.

In order to compute this gradient, Treuille et al. [**125**] augment each simulation state $\mathbf{q}_t$ with its derivative with respect to each control parameter. When estimating the gradient of the objective function and for each frame t, the method updates the derivative of state $\mathbf{q}_t$

according to each control parameter $\mathbf{w}_k$. This costly gradient estimation is performed at each iteration of the quasi-Newton optimization, which becomes prohibitive for long animations simulated in a large domain and involving many control parameters. For this reason, Treuille et al. [**125**] only use a small set of parametric wind and vortex forces. Results obtained using this method can be found in Figure 2.2.1.

Fig. 2.2.1. Five separate 2D simulations (50×50, computation time: $\sim 2-5h$) generated using the method of Treuille et al. [**125**] and forming each letter of the word "SMOKE".

McNamara et al. [**71**] improve on the method to compute the gradient with the adjoint method. The derivative of the objective function with respect to the control vector is given by:

$$\frac{\mathrm{d}\phi}{\mathrm{d}\mathbf{w}} = \frac{\partial\phi}{\partial\mathbf{Q}}\frac{\mathrm{d}\mathbf{Q}}{\mathrm{d}\mathbf{w}} + \frac{\partial\phi}{\partial\mathbf{w}}. \tag{2.2.3}$$

In this equation, matrix $\frac{\mathrm{d}\mathbf{Q}}{\mathrm{d}\mathbf{w}}$ contains a whole state sequence for each control parameter used, which is extremely costly to compute directly. McNamara et al. [**71**] propose to substitute the first term of the sum, considering that:

$$\frac{\partial\phi}{\partial\mathbf{Q}}\frac{\mathrm{d}\mathbf{Q}}{\mathrm{d}\mathbf{w}} \quad \text{s.t.} \quad \left(\mathbf{I} - \frac{\partial\mathcal{F}}{\partial\mathbf{Q}}\right)\frac{\mathrm{d}\mathbf{Q}}{\mathrm{d}\mathbf{w}} = \frac{\partial\mathcal{F}}{\partial\mathbf{w}}, \tag{2.2.4}$$

can be replaced by

$$\mathbf{R}^T\frac{\partial\mathcal{F}}{\partial\mathbf{w}} \quad \text{s.t.} \quad \left(\mathbf{I} - \frac{\partial\mathcal{F}}{\partial\mathbf{Q}}\right)^T\mathbf{R} = \frac{\partial\phi}{\partial\mathbf{Q}}^T, \tag{2.2.5}$$

with $\mathbf{R} = [\mathbf{r_1}, ..., \mathbf{r_n}]$ called the adjoint vector. Reformulating the constraints in the previous equation gives a relationship between each adjoint state $\mathbf{r}_t$ and its previous state $\mathbf{r}_{t-1}$. After computing and storing each state $\mathbf{q}_t$ during a forward pass, the adjoint states are computed in a second reverse pass. The method presented by McNamara et al. [**71**] is orders of magnitude faster than the method of Treuille et al. [**125**], and it allows for a much finer control (i.e., a larger space of control parameters). Note that McNamara et al. [**71**] also extend their method to control free-surface liquids as shown in Figure 2.2.2.

Both Treuille et al. [**125**] and McNamara et al [**71**] require the intermediate solutions at each iteration of the optimization to be a valid solution, which can be overly conservative

Fig. 2.2.2. Controlled liquid (above) and smoke (below) simulations in 3D ($45 \times 50 \times 36$, 46 frames, computation time: $\sim 48h$) generated using the method of McNamara et al. [**71**].

since we only want the final solution to satisfy the constraints imposed by the governing equations. Another type of optimization method, called proximal method, can be applied to tackle this issue. This type of method minimizes an objective function defined as the sum of two convex functions. Each term is approximately and alternatively minimized during an iteration process while convergence is not reached. One of these methods, called the Alternating Direction Method of Multipliers (ADMM) method, has been applied by Pan et al. [**88**] to solve the problem formulated by McNamara et al. [**71**]. The problem is split into two smaller sub-problems, each of them being solved using a different strategy. In the approach from Pan et al. [**88**], a first sub-problem, called Advection Optimization (AO), consists in finding an optimal sequence of velocity fields $\mathbf{u}_t^*$ to advect ρ_t such that it matches the "keyframes" as much as possible, assuming that the $\mathbf{u}_t^*$ are uncorrelated. In practice, it corresponds to fix $\mathbf{u}_t$ and ρ_t, and to solve for $\mathbf{u}_t^*$. A second sub-problem, referred to as the Navier-Stokes Optimization (NSO), enforces the correlation between $\mathbf{u}_t$ given the sequence of $\mathbf{u}_t^*$. In practice, $\mathbf{u}_t^*$ is fixed and we solve for $\mathbf{u}_t$ and ρ_t, considering the Navier-Stokes equations

Fig. 2.2.3. Smoke simulation in 3D ($64 \times 64 \times 64$, 150 frames, computation time: $\sim 26h$) generated using the method of Pan et al. [**88**]. The target changes during the simulation, and goes from letter "A" to "B" to "C".

as the active constraints. These sub-problems are solved alternatively during an iterative process using two different methods. Pan et al. [**88**] use a fixed-point iteration method to solve the AO sub-problem, and the multi-grid Full Approximation Scheme (FAS) to solve the

NSO sub-problem. This technique leads to a major speedup compared to previous methods. Results from Pan et al. [**88**] can be found in Figure 2.2.3.

Another slightly more general proximal method, called Primal-Dual optimization, has been introduced to fluid simulation by Inglis et al. [**49**]. The authors present a general optimization framework that can be applied to several fluid simulation problems. Specifically, the method solves for a new velocity field by minimizing a custom objective function that depends on the problem to solve (e.g., smoke guiding or boundary conditions enforcement in their paper) as well as its divergence. When applied to guide a smoke simulation, an objective function is used, measuring the proximity between the current velocity field and a target velocity field. The optimization occurring in Primal-Dual optimization is parameterized using variables that can be tweaked to improve convergence. For a given nonoptimal set of parameters, Primal-Dual optimization can be reduced to ADMM. Note that contrary to the previous methods [**125, 71, 88**], this work does not perform an optimization over time.
Pan et al. [**86**] develop an interactive sketch-based approach to control a low-resolution liquid simulated with FLIP. After selecting one frame of the fluid simulation, the user can edit the liquid using three control metaphors: sketching strokes, fluid dragging, or local control using a small meshed patch. The resulting control input is used to generate a set of control particles. After choosing a subset of particles between either free surface particles,

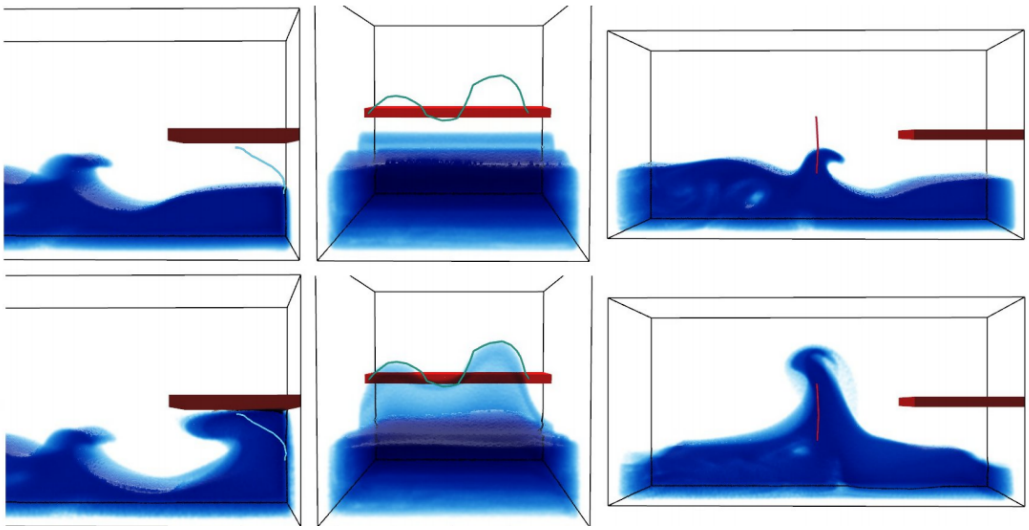


Fig. 2.2.4. Controlled 3D simulations generated with the method of Pan et al. [**86**]. The top row shows frames from liquid simulations as well as input sketched strokes. The bottom row shows the resulting controlled simulations at the same frames after optimization, with a control applied to (from left to right) solid boundary particles, free surface particles, and medial axis particles.

solid boundary particles, and medial axis particles, an optimization algorithm solves for local parametric forces minimizing the difference between both generated control particles and a subset of particles selected by a user. Once computed, the center of each force is back-advected before applying the force to the simulation with a smooth falloff in order to propagate the changes to the previous time steps. Figure 2.2.4 shows control applied on different subsets of liquid particles using this technique.

Methods performing a space-time optimization to drive a fluid to match a sparse set of user-defined targets [**125, 71, 88**] scale poorly, and therefore, do not appear like a reasonable option to control large and complex simulations often used in the animation or VFX industries. Another drawback of this type of approach is that keyframes depicting a plausible complex liquid or smoke system are usually hard and complex to produce for an animator.

However, these methods usually depend on a small set of parameters, are robust, and usually minimize the overall control applied to a simulation. Pan et al. [**86**] reduce the size of the problem, both temporally (i.e., by performing a numerical optimization for one frame only) and spatially (i.e., by considering only a subset of particles in the optimization), providing an interactive control experience for low-resolution simulations. Although these simplifications lead to more approximate overall control, this strategy remains an attractive direction for future research.

2.2.2. Empirical Forces

Controllers using empirical forces have been proposed to control fluid simulations. Some of these controllers can be seen as proportional controllers. At each simulation time step, such a controller estimates an error between the current fluid state and a user-defined target. A responsive correction is applied to reduce this error, usually by applying external control forces to the system.

In order to drive a smoke simulation to match a target, Hong and Kim [**45**] propose to add an extra dimension, called potential, to the simulation domain. Potential values are initialized in the target (at a value K_{target}), on the boundary of the simulation domain (at a value $K_{boundary}$), and in the region corresponding to the initial smoke distribution (at a value $K_{initial}$), such that $K_{target} < K_{initial} < K_{boundary}$. Potential values in the remainder of the simulation domain are found by solving the Laplace's equation $\nabla^2 K = 0$, with the Dirichlet boundary condition previously defined. The gradient of the potential field K is then computed and stored in the simulation grid. For a given static target, note that these steps have to be performed only once and can be precomputed. During simulation, a force

$F(\mathbf{x}) = -\nabla K(\mathbf{x})$ is applied at each simulation time step, driving the smoke from regions with high potential values to regions with low potential values (i.e., target) as shown in Figure 2.2.5. This method is simple and adds nearly no overhead at run-time when used with a static target. However, this strategy struggles to reproduce thin features and has only been applied to smoke simulation.

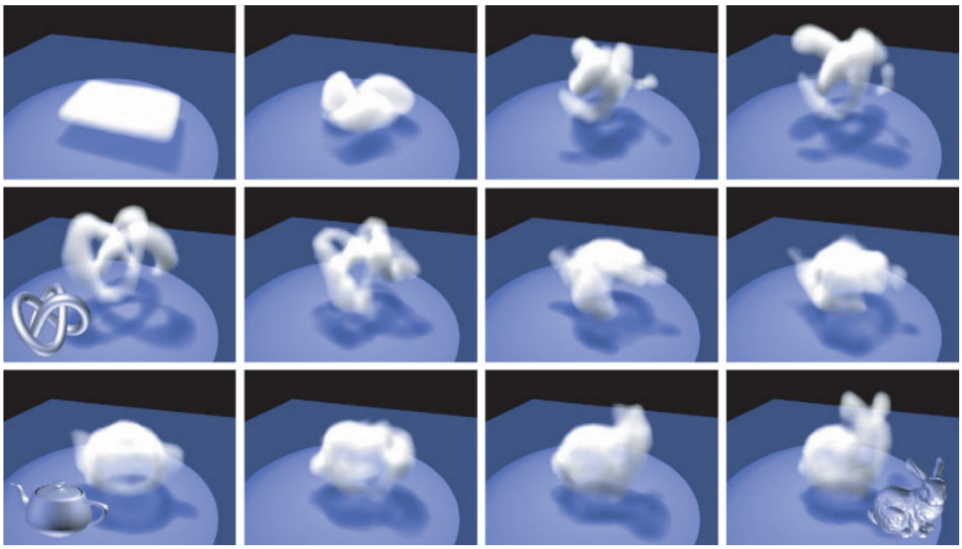


Fig. 2.2.5. Controlled smoke simulation from Hong and Kim [**45**]. Targets are changed during the simulation, going from a torus knot, to a Utah teapot, to a Stanford bunny.

Fattal and Lischinski [**30**] propose to use a driving force to transport smoke density to match a target. This force is defined as:

$$F(\rho, \rho^*) = \tilde{\rho}\frac{\nabla(\tilde{\rho}^*)}{\tilde{\rho}^*}, \tag{2.2.6}$$

with ρ the density field of the smoke simulation, ρ^* the target density field, $\tilde{\rho}$ a blurred version of ρ, and $\tilde{\rho}^*$ a blurred version of ρ^*. In addition to this force, a smoke gathering term is introduced to counteract numerical dissipation that causes the smoke to diffuse. This gathering term is given by:

$$G(\rho, \rho^*) = \nabla \cdot \left[\rho\tilde{\rho}^*\nabla(\rho - \rho^*)\right]. \tag{2.2.7}$$

This approach is simple and efficient, adding only a small overhead to the total simulation time. However, and as pointed out by the authors, the use of a gathering term $G(\rho, \rho^*)$ leads to "ghosting" artifacts: the target shape seems to emerge from an amorphous cloud of smoke, resulting in strange and unnatural behaviors of smoke as illustrated in Figure 2.2.6.

Fig. 2.2.6. Smoke simulation controlled to match a moving mouse target using the strategy introduced by Fattal and Lischinski [**30**].

Using implicit functions to represent both the smoke region and the target (static or dynamic), Shi and Yu [**111**] drive a smoke simulation so that a specific isosurface of the density field matches the zero level set of a target object. At each time step, feedback forces are applied to the simulation to reduce the amount of discrepancy. The control strategy is divided into two stages: a first stage where the smoke has to reach the target, and a second stage where the smoke has to keep track of the evolution of the target. By applying a morphing technique between the initial smoke distribution and the target, both these stages can be treated with the same method. During the simulation and at each time step, a small-motion tracking is performed. A large-motion tracking is also performed if the amount of smoke overlap between two consecutive frames is below a given threshold. Specifically, small-motion tracking consists in iteratively pushing in and/or pulling out the smoke boundary along its local normal to minimize a matching metric. The large-motion tracking directly transports the smoke from its previous location to its second location. Results of smoke controlled to match a target mesh can be found in Figure 2.2.7.

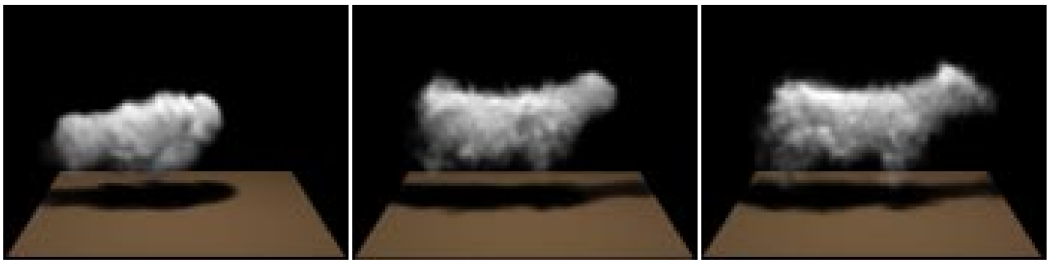


Fig. 2.2.7. Example of smoke controlled using the method of Shi and Yu [**111**] to match a cow mesh target.

In order to control a liquid to match a rapidly changing target and produce animations such as in Figure 2.2.8, Shi and Yu [**112**] apply two external forces to the simulation. A first

feedback force minimizes both the velocity and shape discrepancies between the simulated liquid and the target. The velocity component is trivial and is defined as $\mathbf{f}_{vel} = -\beta(\mathbf{u}_L - \mathbf{u}_T)$. The shape component is more complex to define. First, the value of the force is defined on the liquid boundary. At a point $\mathbf{x}$ located in the target shape, the force has to push the liquid to the outside and is set to $\tilde{\mathbf{f}}_{shape} = -\alpha d_T \frac{\nabla d_T}{||\nabla d_T||}$, with d_T the signed distance function representing the target. If the point is located outside of the target, the force is set to $\tilde{\mathbf{f}}_{shape} = -\alpha d_L \frac{\nabla d_L}{||\nabla d_L||}$, with d_L the signed distance function representing the liquid. In both cases, this force pulls the liquid toward the target boundary. The final expression of the shape force on the liquid boundary is obtained by removing the divergence component of the field, ensuring a null total flux through the boundary. Note that this type of force has also been applied by Shin and Kim [**113**] at the interface between the phases of a multiphase fluid during the projection step. To calculate the shape feedback force in the remaining regions of

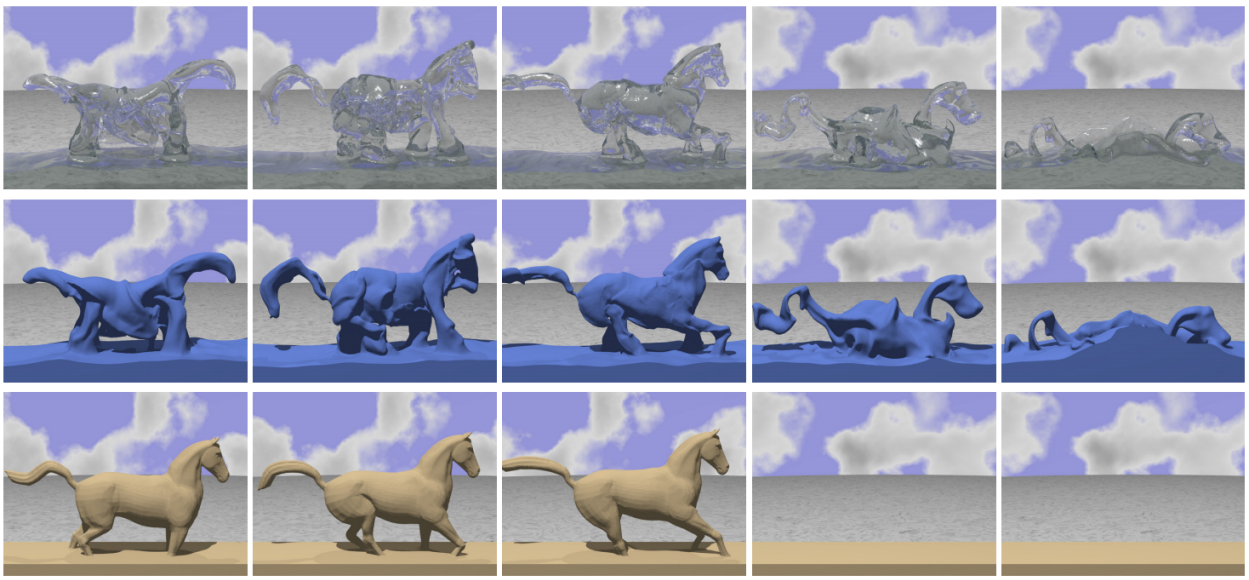


Fig. 2.2.8. A liquid is controlled using the method of Shi and Yu [**112**] to match an animated mesh representing a galloping horse. Top and middle row: Surface of the controlled liquid, rendered using respectively a water-like and a Lambertian shading. Bottom row: The animated target used to control the liquid simulation.

the simulation space, Shi and Yu [**112**] define the force field as the gradient of a scalar field H. The Laplace equation $\nabla^2 H = 0$ is then solved, with boundary conditions at the liquid boundary corresponding to the forces previously defined. Once computed, the gradient of the scalar field H gives the shape component of the first feedback force. The second feedback force used corresponds to the gradient of a potential defined using the shape and the skeleton

of the target. This force is similar to the one introduced by Hong and Kim [**45**], but the potential field used is a monotonically increasing function of the signed distance of the target shape. This potential is given by:

$$\phi(\mathbf{x}) = C\,\mathrm{sgn}(d_T(\mathbf{x}))\,|d_T(\mathbf{x})|^{\gamma}\,, \tag{2.2.8}$$

with γ and C user-defined parameters. The potential field used leads to an increasing magnitude of the gradient when the distance to the target surface becomes larger, leading to a larger control force for liquid located further away. Inversely, the control is reduced when the liquid is close to the target, leading to more natural fluid motion at the boundary of the target. Note that inside the target, the force field points to the central skeleton of the target shape. It differs from Hong and Kim [**45**] for which the force field would be null in that case.

Stuyck and Dutré [**119**] introduce a sculpting tool for particle-based liquids. The editing process is divided into two phases illustrated in Figure 2.2.9. During a manipulation phase, the liquid is sculpted by a user and the resulting particle distribution is stored as a keyframe. Note that a relaxation scheme is applied on the particle distribution in order to obtain a more plausible liquid configuration. This keyframe is then used during a control phase, during which forces are computed and used to reduce the distance between each liquid particle and its corresponding target.

Fig. 2.2.9. Sculpting a liquid simulation using the approach from Stuyck and Dutré [**119**]. From left to right: An initial liquid simulation at a user-defined frame as well as user intents represented with red arrows, a sculpted keyframe after relaxation, the resulting liquid controlled to match the relaxed keyframe.

Zhang et al. [**139**] introduce a method to control a liquid simulation using an animated skeleton such as in Figure 2.2.10. A user first provides an initial skeleton mesh associated with an initial velocity field as well as keyframes describing an animated skeleton. The system also extracts a skeleton from the initial liquid distribution before transforming these data into a skeleton description that includes keypoints, bones, and clusters (e.g., fluid volumes). Given a current skeleton description (called source) and a target skeleton, a matching problem is solved, finding the flow matrix describing the volume of liquid to transfer from each source cluster to each target cluster. Note that the algorithm solves for one-to-one relationships

between clusters. To do so, the system does cluster separation or merging if needed. During a control phase, feedback forces are used to drive each cluster towards its target, minimizing both the distance between centers of mass and velocities. Skeleton rigidity constraints are applied on the velocity field in order to make it tightly follow the motion of the skeleton when close to the bones.

Fig. 2.2.10. Liquid controlled to match an animated skeleton using the method of Zhang et al. [**139**]. The four leftmost images show poses of a skeleton at four different times. The remaining four images represent the liquid surface controlled using the skeleton, at times corresponding to these poses.

Kim et al. [**61**] propose to control smoke simulations using NURBS curves forming a path, such that the smoke is driven along the path. At each frame, the system evaluates the difference between the velocity field $\mathbf{u}$ obtained by simulation and a procedurally generated velocity field $\mathbf{U}$ implied by the path. To generate this procedural velocity field $\mathbf{U}$, the path is first segmented in a set of overlapping portions such that each portion does not self-intersect nor contain high curvature. At each frame, a navigation control module determines the portion of the path that is currently used to control the smoke based on the fluid front. This section of the path is then used to generate a target velocity field by combining four components. A first tangent field, defined at a distance smaller than the influence radius R to the curve, drives the smoke along the path. A Rankine velocity field is created to produce a bulk rotational motion around the path, in a plane whose normal is oriented along the path. A normal field pushes the smoke toward the path. Finally a vortex field is used to inject small-scale swirling motions to enhance the visual aspect and plausibility of the controlled smoke simulation. The combination of these fields becomes the target velocity field $\mathbf{U}$, thereafter used to apply a feedback control by using the following force on the simulated smoke:

$$\mathbf{f}_t = \mathbf{F}_b - \gamma(\mathbf{u} - \mathbf{U}), \tag{2.2.9}$$

where $\mathbf{F}_b = \frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} - \eta \nabla^2 \mathbf{U}$ measures the body force generated by the target flow. $\mathbf{F}_b$ can be seen as an approximation of the force required to make $\mathbf{u}$ equal to $\mathbf{U}$, while $\gamma(\mathbf{u} - \mathbf{U})$

is a damping term. An example of smoke controlled using the method of Kim et al. [**61**] can be found in Figure 2.2.11.

Fig. 2.2.11. Example from Kim et al. [**61**] in which smoke simulations are controlled with NURBS curves to form each letter of the words "SURRENDER DOROTHY".

Patel et al. [**89**] use an interpolation technique to generate a divergence-free field from user-defined data generated on a uniform lattice of voxels. Once generated, the target velocity field is used to apply feedback forces on a particle system.

Yang et al. [**135**] introduce a framework that unifies path-based and shape-based control. Signed distance fields (SDF) computed from the input geometries and paths (represented as NURBS) are used to generate three types of control forces. Note that each type of force uses a specific SDF. Path control forces (SDF: $\phi_{path}(\mathbf{x}) = d_{curve}(\mathbf{x}) - R$, with R the radius of influence) drive the smoke along a path. Shape control forces (SDF: $\phi_{shape}(\mathbf{x}) = d_{surface}(\mathbf{x})$ if $\mathbf{x}$ is outside the shape, $\phi_{shape}(\mathbf{x}) = -d_{surface}(\mathbf{x})$ otherwise) drive the smoke to form target shapes. Finally, boundary control forces (SDF: $\phi_{mixed} = \min(\phi_{shape}, \phi_{path})$) pull the smoke located outside any path or shape to the nearest target region. Therefore, this latter force is only active in regions where the SDF $\phi(\mathbf{x})$ is positive. It is defined as:

$$\mathbf{f}_{boundary}(\mathbf{x}) = \rho \frac{|\phi(\mathbf{x})|}{\Delta x} \left(-\frac{\nabla \phi(\mathbf{x})}{||\nabla \phi(\mathbf{x})||} \right), \tag{2.2.10}$$

where $\phi(\mathbf{x})$ can be either $\phi_{path}(\mathbf{x})$, $\phi_{shape}(\mathbf{x})$, or $\phi_{mixed}(\mathbf{x})$. Δx is the length of the grid cell, and ρ is the smoke density. Note that the further the smoke from the target, the stronger the boundary control force. Inside a target shape, shape control forces are emitted from a medial axis point cloud and are defined as:

$$\mathbf{f}_{medial}(\mathbf{x}) = \rho S \left(\frac{d_p(\mathbf{x})}{R_w} \right) G(\mathbf{x}), \tag{2.2.11}$$

Fig. 2.2.12. Example from Yang et al. [**135**] where a combination of path-based and shape-based controls is used to drive a smoke along a path until reaching a region where the smoke is controlled to match a target mesh.

with S a step function, $d_p(\mathbf{x})$ the distance from $\mathbf{x}$ to the medial axis, R_w the influence radius of the medial axis, and ρ the smoke density. $G(\mathbf{x})$ returns a unit vector pointing towards the medial axis point location. Finally, the path control force, active at a distance shorter than R_f from a path, is defined as:

$$\mathbf{f}_{path}(\mathbf{x}) = \rho S \left(1 - \frac{d_c(\mathbf{x})}{R_f}\right) T(\mathbf{x}), \tag{2.2.12}$$

with $d_c(\mathbf{x})$ the distance from $\mathbf{x}$ to the curve, and $T(\mathbf{x})$ a unit vector at $\mathbf{x}$ tangent to the path. A hybrid vortex particle scheme, combining the properties of vortex particles and Langevin particles, is also used to enhance turbulence flow details. These vortex particles are used to compute a vorticity force that is used as an additional external force to the system. A control scenario using both path-based and shape-based control is illustrated in Figure 2.2.12.

After generating a target velocity field called "influence field" from primitives created in the scene by a user (e.g., streamtube flow primitives), Gates [**37**] linearly blends the intermediate velocity field (after self-advection and application of the forces, and before projection) of a smoke simulation with the target velocity field. Note that the blending factor is a spatially and temporally varying user-defined parameter.

Using a description of a fluid flow based on harmonic analysis, Angelidis et al. [**4**] introduce a framework to generate controllable vortex-based smoke simulations such as in Figure 2.2.13. Each vortex filament primitive is described using a local coordinate system computed using principle component analysis, and three Fourier series. This representation

is meaningful to a user: the local coordinate system gives both an average position of the filament and a direction of the motion, while the harmonic coefficients give information about the complexity of the filament. In order to control smoke, two operations are provided: pad-

Fig. 2.2.13. Three vortex-based smoke simulations guided with different control curves using the method introduced in Angelidis et al. [**4**].

dling redistributes the filament strength such that it induces a wind in a desired direction, and turning reorients the support of the function. High-level control tools are provided to users to modify the large-scale motion of the filaments. First, they can be controlled using a control curve. An arc-length parameter along the curve is defined and updated while the filament is updated. This parameter is used to estimate the new direction (i.e., the tangent at the control curve) that the filament should take at each simulation time step. Attractors can also be used to make the filaments paddle towards them. Finally, tornado-like effects can be achieved by making the filaments spin around the vector along which they move. Note that in addition to high-level controls, small-scale perturbations can be added to enrich the simulation with visual details.

The methods described in this section are usually efficient because the control to apply on a chunk of fluid can be usually computed by simply evaluating a function. In most cases, such a function estimates the distance between the current state and a target, and returns a feedback correction (a velocity correction or a force) to bring the fluid state closer to the current target state. However, these methods often rely on several user-defined parameters that should be adjusted depending on the characteristics of the scene.

2.2.3. Control Particles

In another form of control strategies, control particles have been used to locally change the behavior of a fluid. This section describes the major contributions in this category.

Thürey et al. [**124**] propose to generate control forces in the neighborhood of special particles used to drive a liquid. These control particles are generated using either a pre-computed function, by sampling a target shape, or by using another fluid simulation. They are not rendered nor simulated, but instead generate control forces on the liquid particles in the scene. Two types of forces are used in this method. First, attraction forces bring an element of fluid e closer to control particles i that are not covered by enough liquid. This attraction force is defined as:

$$\mathbf{f}_a(e) = w_a \sum_i \alpha_i \frac{\mathbf{x}_i - \mathbf{x}_e}{||\mathbf{x}_i - \mathbf{x}_e||} W(||\mathbf{x}_i - \mathbf{x}_e||, h), \tag{2.2.13}$$

with $\alpha_i = 1.0 - \min(1, \sum_e V_e W(||\mathbf{x}_i - \mathbf{x}_e||, h))$ a factor reducing the attraction of a control particle when it is sufficiently covered with liquid. Second, and in order to give the liquid a motion similar to the motion of the control particles, a velocity force is also applied, which minimizes the difference between the velocities of both liquid and local control particles. The force is defined as:

$$\mathbf{f}_u(e) = w_u \sum_i (\mathbf{u}_i - \mathbf{u}_e) W(||\mathbf{x}_i - \mathbf{x}_e||, h). \tag{2.2.14}$$

Note that this control is only applied on the low-frequency component of the velocity field to preserve the small-scale fluid-like details of the simulated liquid. A liquid controlled using this strategy is shown in Figure 2.2.14.

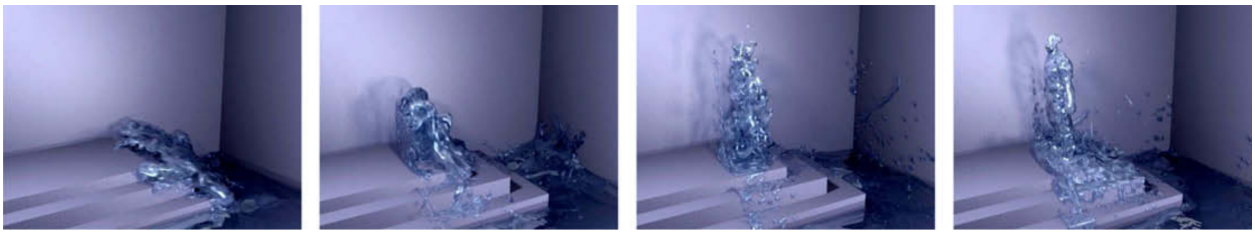


Fig. 2.2.14. Controlled fluid character generated using the method of Thürey et al. [**124**].

Madill and Mould [**69**] use particles to drive smoke (simulated using marker particles) to match a target shape such as in Figure 2.2.15. After sampling both the smoke and the target with N particles, a pairing is generated between smoke and target particles. This pairing is defined as an approximation of an optimal solution minimizing the sum of distances from each particle located in the smoke to its counterpart in the target. These pairs are used to generate attraction forces that are interpolated and stored in a uniform grid. In order to enrich the dynamics of the smoke with turbulent motion, the authors use vortex particles to generate a turbulent velocity field. At each position in the simulated smoke, a velocity-from-vorticity equation is used to estimate the velocity generated by the neighboring vortex

particles. This equation is used to estimate and store the velocity generated by vortex particles in a uniform grid. Vortex particles and smoke marker particles are finally advected through the combined velocity field.

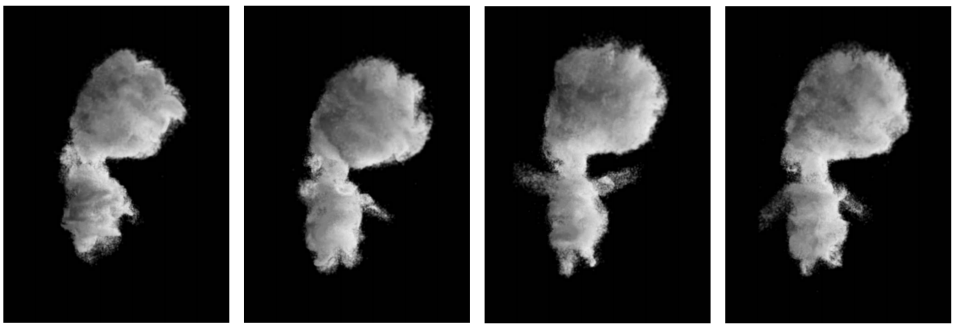


Fig. 2.2.15. Smoke simulation controlled using the method of Madill and Mould [**69**] to match an animated creature.

Rasmussen et al. [**95**] use control particles to generate soft and hard time-varying constraints on a liquid. Four types of control particles are defined, based upon the physical quantity (i.e., velocity, viscosity, level set, and velocity divergence) to control. In their system, each particle locally modifies the properties of the fluid simulated. This control depends on the shape associated to a control particle (e.g., a sphere or a cylinder) as well as a falloff according to a distance to it. For a soft constraint, the falloff curve gives an

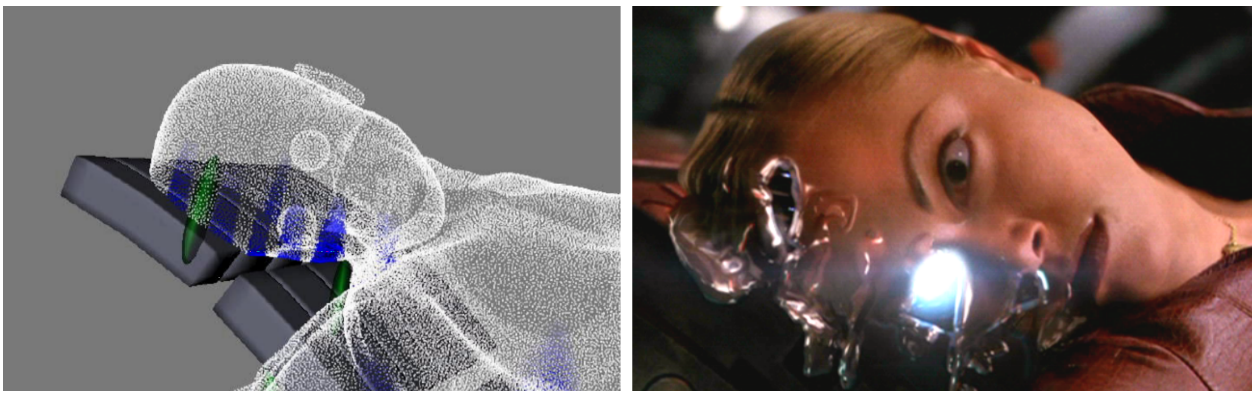


Fig. 2.2.16. Control applied on a metallic liquid simulation using the method of Rasmussen et al. [**95**]. Left: Control particles in white apply hard velocity constraints. Control particles in blue apply soft velocity constraints, leading to an effect of melting and flowing. Green regions represent CSG (Constructive Solid Geometry) level set liquid erasers. Right: Final rendering of the scene.

interpolation parameter α to compute a new physical quantity V as a linear interpolation:

$V_{new} = (1-\alpha)V_{liquid} + \alpha V_{control}$. Conversely, a hard constraint directly fixes a physical quantity corresponding to the attributes carried by the local control particles. A rendering of a metallic liquid simulation controlled using this technique can be found in Figure 2.2.16.

Lu et al. [**67**] propose to extend the concept of skeletal animation to control a fluid simulation. During a first fluid rigging step, an animated point cloud is generated. Rigid-body motion as well as incompressible deformations can be applied on the point cloud in order to generate a target animation. The simulated fluid is then controlled to match the user-specified motion as shown in Figure 2.2.17. Specifically, each rig point exerts a virtual drag force on the liquid particles located in its neighborhood. Note that, since the incompressibility condition is strictly respected by the rig, the velocity of the fluid controlled with the drag force is guaranteed to be a plausible flow field. Instead of directly using this velocity field, an

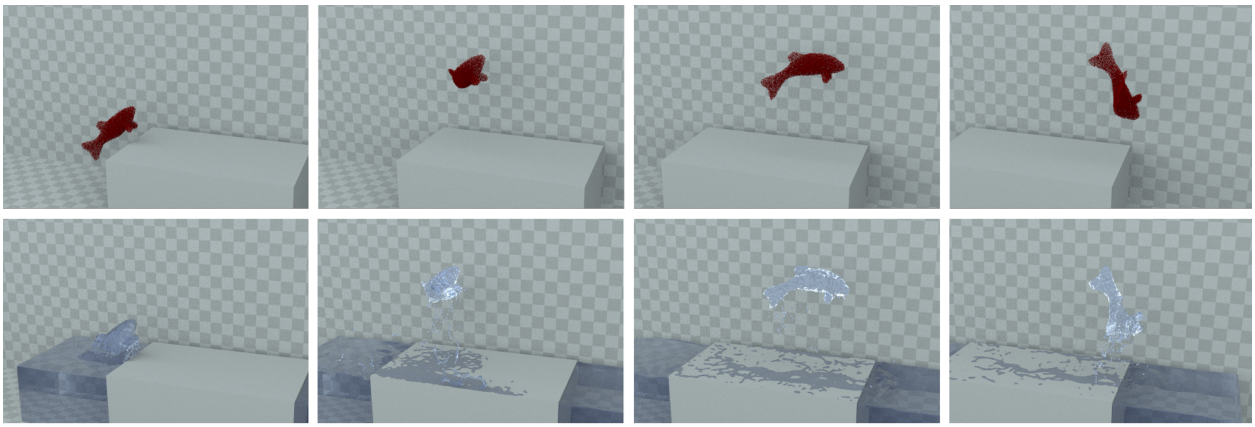


Fig. 2.2.17. Liquid controlled with a rig point cloud using the method of Lu et al. [**67**].

optimization problem is solved during what the authors call a linear flow skinning step. This optimization problem solves for a velocity field $\mathbf{u}$ containing the same zero-order (velocity) and first-order (velocity gradient) features than the velocity field that we generated $\mathbf{u}_{temp}$. This optimization problem is defined as:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \lambda_1 \left|\left|\nabla\mathbf{u} - \nabla\mathbf{u}_{temp}\right|\right|^2 + \lambda_0 \left|\left|\mathbf{u} - \mathbf{u}_{temp}\right|\right|^2. \tag{2.2.15}$$

This control strategy has been applied to both liquid and smoke, simulated using either a particle- or grid-based fluid solver.

After automatically generating control particles by sampling a target shape, Zhang et al. [**140**] provide a position-based control technique to make a controlled fluid match the target as in Figure 2.2.18. More precisely, three constraints are added and solved for under a position-based framework: (1) a density constraint to maintain a target liquid density, (2) a

spring constraint that takes effect only when a fluid particle is far from its corresponding control particle to match fast moving targets, and (3) a velocity constraint used to modify the flow of fluid.

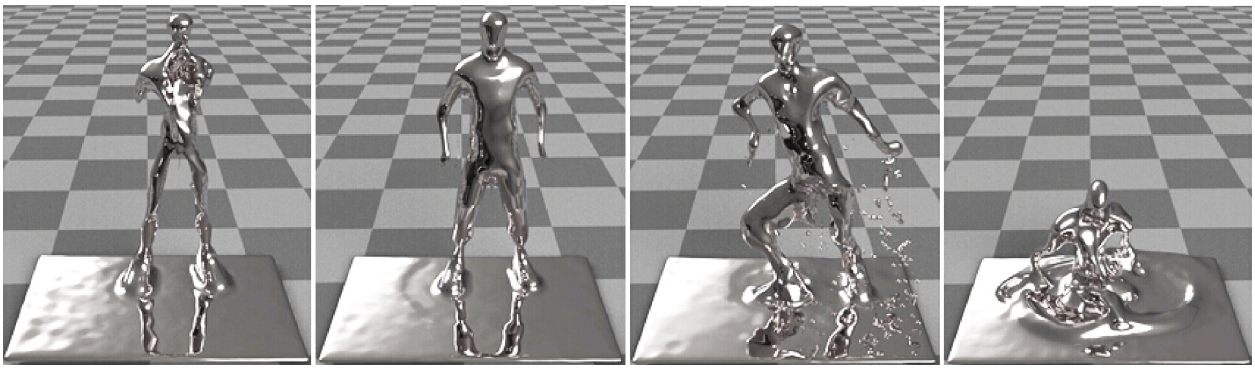


Fig. 2.2.18. Dancing character generated using the position-based control method introduced by Zhang et al. [**140**].

In addition to be easy to integrate in a particle-based fluid solver, methods based on control particles usually scale well since a parcel of fluid is only influenced by the control particles in its neighborhood. One difficulty of this type of technique is to transfer the target to match (i.e., skeleton, mesh, sketch) into a set of control particles, rising the question of how dense the target sampling should be.

2.2.4. Enforcement of Boundary Conditions

Some fluid control strategies consist in imposing Neumann boundary conditions during the pressure solve step. This strategy is similar to defining boundary conditions at a liquid-solid interface, except that both the regions of space and velocity information used to fix these boundary conditions are defined by the user.

Raveendran et al. [**96**] rasterize control meshes on the simulation grid to control a liquid. A sparse set of user-defined keyframes is interpolated by a volume-preserving morphing technique to derive a dense sequence of target shapes. The velocity at the surface of the target, computed using finite differences between successive frames, is transferred on the grid cell faces and used as Neumann boundary conditions in the pressure Poisson equation of a liquid simulation. To create some secondary motions in the liquid simulation, Raveendran et al. [**96**] propose to relax the control applied on the system by using a linear interpolation between the velocity obtained by imposing these Neumann boundary conditions at the surface of the mesh and the velocity without any of these boundary conditions. Relaxing control

tends to spill out of the control shapes. In order to tackle this issue, the authors make the velocity field locally divergent in order to compensate for the lost volume of liquid. Note that a Lagrangian surface tracker is used in order to represent extra details (such as ripples) that can be thinner than a grid cell. A liquid character generated using this technique is shown in Figure 2.2.19.

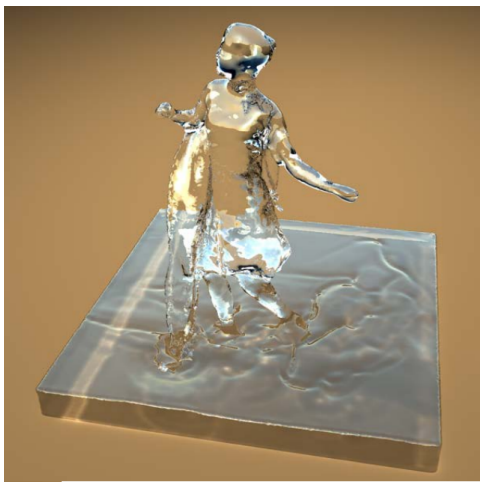


Fig. 2.2.19. Liquid controlled to form a dancing character using the method of Raveendran et al. [**96**].

Fig. 2.2.20. Fluxed Animated Boundary from Stomakhin and Selle [**118**] used to create an art-directed simulation.

Based on a similar idea, Stomakhin and Selle [**118**] introduce the Fluxed Animated Boundary (FAB) method, an artist-friendly control tool used in some shots of Disney's feature film *Moana* [**25**]. Specifically, this tool can perform control over a FLIP liquid simulation by adding two extra steps to the simulation algorithm. First, the method enforces boundary conditions in the Poisson projection, in regions close to the surface of the FAB. The material velocity field sampled to estimate these values is obtained either by using an analytical expression, by locally estimating the velocity of the flow, or by directly painting the material flow for more art-directed results. Second, a seeding method samples extra particles in a thin layer around the control shape, whose thickness depends on the CFL condition. This reseeding, performed before advection, guarantees that new particles are present to be advected on the border of the control shape in the case of a locally outgoing flow. All the particles localized inside the FAB after advection are deleted. The resulting method has been used to generate both art-directed (e.g., creating an art-directed swirling stream of water such as in Figure 2.2.20) and realistic scenarios (e.g., seamlessly integrating a FLIP simulation into a procedural ocean).

Imposing custom boundary conditions during the pressure solve step of a fluid simulation is an elegant solution to apply control on a simulation. This technique has been applied using physically-based velocities, extracted from a low-resolution model [**81, 85**] or from a procedural model [**118**], respectively to simulate a high-resolution liquid on a layer surrounding a low-resolution liquid, or to simulate a region of liquid on top of a procedural ocean. In all these examples, boundary conditions are set in a thin layer surrounding a control region, but they lead to velocities everywhere in the domain that respect these control velocities. This type of approach is both elegant, natural, and produces great results. When used with a non-physically correct velocity field, extra work is necessary to produce satisfying fluid dynamics, such as relaxing control [**96**] or increasing its visual complexity using noise functions [**118**].

2.2.5. Manipulation of the Simulation Domain

A few techniques propose to directly exploit the grid used to simulate a smoke animation as a user-friendly support to deform the shape of the fluid.

Pan and Manocha [**87**] modify the velocity field of a fluid simulation by deforming the underlying grid using mesh editing tools. In their examples, a simple drag-point control method is used. After modifying the position of a control handle, the global deformation function $\phi(\mathbf{x})$ is computed by minimizing an energy function that depends on the Green's strain of the mesh (i.e., the simulation grid). This deformation is then transferred back

Fig. 2.2.21. Controlled smoke simulations in 2D from Pan and Manocha [**87**]. A simulation is performed on a regular uniform grid (leftmost image). Simulations are generated using the same initial conditions but with three deformed grids.

to smoke animation by transforming the velocity in the advection operator $S\mathbf{u} \cdot \nabla$, with S the component of the deformation gradient $F = \frac{\partial \phi}{\partial \mathbf{x}}$ corresponding to a non-zero elastic

energy and obtained using polar decomposition. The simulation is therefore still executed on a regular nonmodified simulation grid, the transformed grid being used to estimate S, and as a tool to visualize the user transformation. In practice and in order to prevent the modification to be affected by numerical dissipation of the semi-Lagrangian operator, this modification is implemented as an additional force $\mathbf{f} = ((I - S)\mathbf{u} \cdot \nabla)\mathbf{u}$ applied to the system. Note that an additional projection is used to enforce the $\nabla \cdot \mathbf{u} = 0$ property. Also note that the authors improve the performance of the method by restricting the configuration space to a reduced-dimensional subspace. Examples of deformed smoke simulations obtained with this technique can be found in Figure 2.2.21.

Sato et al. [**103**] propose a method to deform a fluid flow field while preserving incompressibility. The method computes a vector potential Φ from an input velocity field by solving a Poisson equation with appropriate boundary conditions. This potential field can

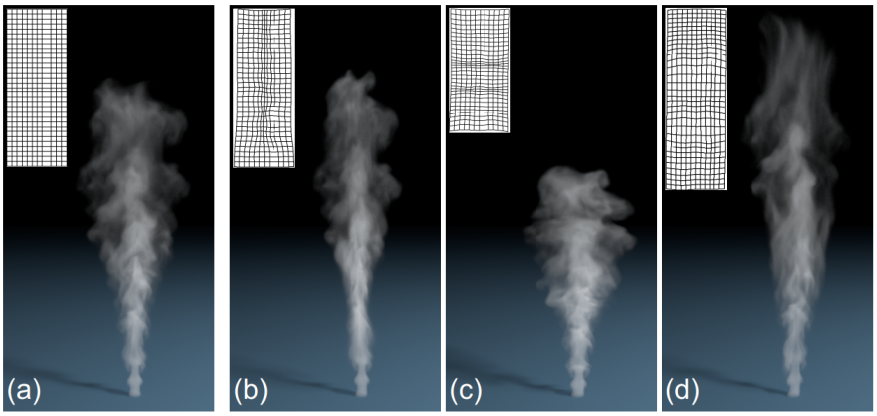


Fig. 2.2.22. Simulations in 3D edited by manipulating the simulation grid using the method of Sato et al. [**103**]. (a) A simple smoke simulation is generated. (b) to (d) A user can produce modified simulations by deforming the simulation grid.

then be user-edited by directly modifying the grid, thus generating a new vector potential $\bar{\Phi}$. The deformed velocity field is simply generated by computing the curl of $\bar{\Phi}$, ensuring its zero-divergence. Results obtained with the method of Sato et al. [**103**] can be found in Figure 2.2.22.

Overall, these techniques should be used when an artist wants a smoke simulation to slightly trend in a different manner than the one obtained without any control. They offer an interface familiar to artists (i.e., mesh with handles) to manipulate the simulated fluid.

2.3. Editing a Fluid as a Post-Process

Instead of controlling a fluid during its simulation, some editing techniques propose to art-direct an already simulated fluid. Note that after editing, either a full [**107**], a partial [**13**], or no fluid resimulation [**92, 70**] are required to obtain the edited animation.

Schpok et al. [**107**] present a system to art-direct smoke simulations by extracting high-level flow features that can be manipulated (i.e., rotated, rescaled, etc.) by the user before resimulating the animation. Manteaux et al. [**70**] propose to extract space-time features on a liquid surface mesh by using topological operations during a semi-automatic process. After a user selects a region on a liquid surface, an automatic process computes the spatial and temporal ranges of the selected feature. Once extracted, these features can then be copied, transformed, duplicated, and pasted anywhere on the same or on another liquid surface, creating a new art-directed liquid surface as shown in Figure 2.3.1. The method is efficient and user-friendly, but can be too limited to generate complex liquid simulations.

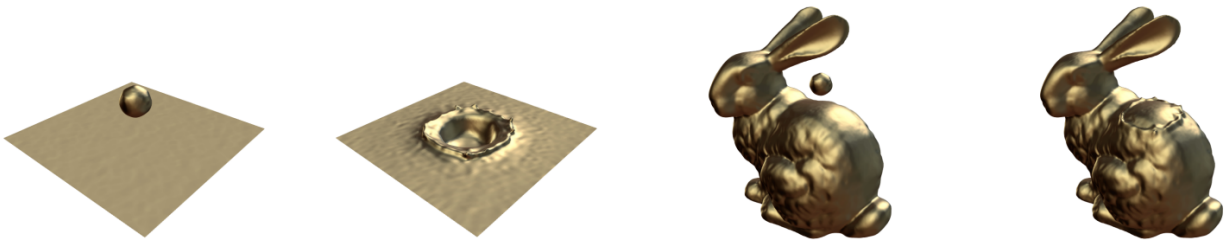


Fig. 2.3.1. Space-time features describing a falling drop of water (two leftmost images) are extracted and pasted onto a Stanford bunny mesh (two rightmost images) using the system of Manteaux et al. [**70**].

Pighin et al. [**92**] introduce the Advected Radial Basis Function (ARBF), a fluid representation that uses particles advected along path lines to parameterize a fluid in an Eulerian simulation. In this model illustrated in Figure 2.3.2, each particle is the center of a Radial Basis Function (RBF) that is used to induce a scalar field (e.g., density, temperature) in its neighborhood. After simulation, the ARBF representation allows the user to edit a fluid by considering these particles as a set of handles that can be manipulated as deformable objects. After a modification and in order to maintain continuity in the flow, spatial and temporal constraints are applied on the system. The spatial constraints are enforced by using an SPH framework in order to iteratively compute pressure forces and integrate each particle forward in time until reaching an equilibrium. The temporal constraints are guaranteed by using hierarchical B-Splines interpolation in order to propagate the modifications forward and backward in time. Since this interpolation does not take into account the physics of the

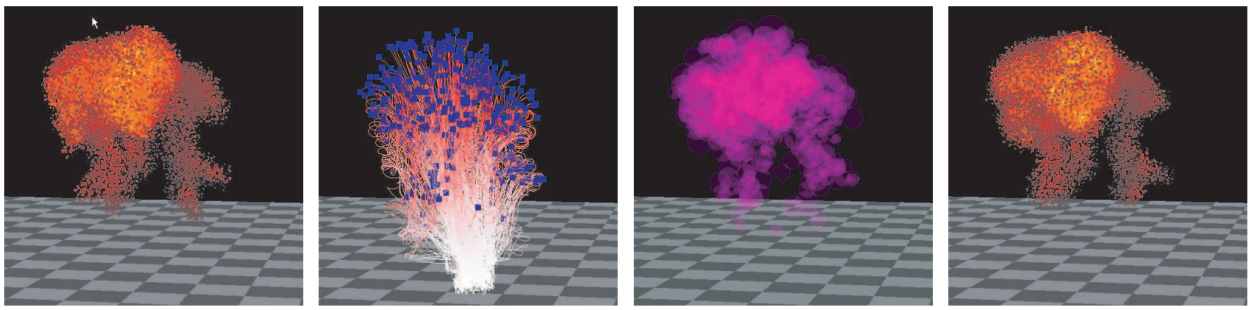


Fig. 2.3.2. ARBF fitting from Pighin et al. [**92**]. From left to right: An initial fluid simulation, pathlines of the simulated particles, an RBF is fitted for each particle, and sampling of the ARBF model.

fluid, it could likely lead to artifacts. This method has been tested on buoyancy-driven flows such as explosions, rising smoke stacks, steam jets, and fires.

Bojsen-Hansen and Wojtan [**13**] propose a method to seamlessly integrate a fluid simulation into a larger scenario, allowing an artist to locally resimulate a portion of an already simulated liquid as shown in Figure 2.3.3. To do so, their method uses generalized nonreflecting boundary conditions that damp the outgoing waves in a thin layer at the borders of the simulation domain, as they exit the liquid region inserted in the larger simulation. Specifically, this method extends the concept of Perfectly Matched Layers (PML). After

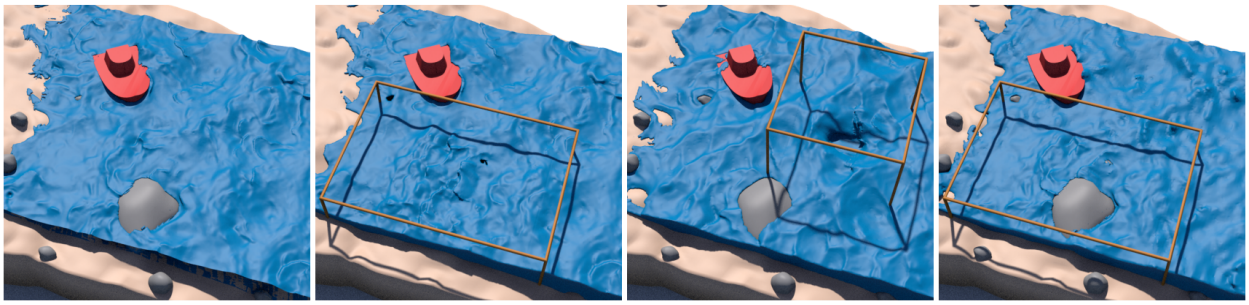


Fig. 2.3.3. The method introduced by Bojsen-Hansen and Wojtan [**13**] can be used to make local changes retroactive in a simulation. From left to right: An input liquid simulation, the same simulation edited by removing the rock in the water as well as the resulting liquid response, by adding a splash, or by simulating a higher resolution liquid simulation.

transforming the governing equation to the frequency domain using Fourier transform, a spatial stretching is applied to damp waves traveling in the boundary layer. Instead of damping the velocity field to zero in the boundary layer, Bojsen-Hansen and Wojtan [**13**] propose to damp it to the velocity of a background flow in order to integrate the simulation in an already simulated liquid. This is done by defining the velocity field as the combination

of a background flow and a perturbation flow, and by damping the perturbation component to zero. In order to deal with cases where the velocity field would generate entirely new liquid surface geometry (e.g., splashes) in the simulation domain, the liquid geometry from the background is copied within a layer whose thickness depends on the CFL condition. Note that this method is not compatible with simulation techniques commonly used in the industry (e.g., FLIP, semi-Lagrangian advection), suggesting a good avenue of future work.

Techniques to edit a fluid after simulation are of interest, especially to perform minor modifications (e.g., a correction on an already simulated model). Methods that do not perform a resimulation [**92, 70**] after editing might struggle to produce satisfying results when a user wants to make more drastic changes on a fluid. The strategy proposed by Bojsen-Hansen and Wojtan [**13**] is elegant. By performing a resimulation only in a small region of space, the method allows an artist to localize his editing as well as to save computation time by keeping the rest of the liquid as is.

2.4. Fluid Interpolation

Some methods propose to reuse simulation data in order to generate new fluid animations, either by interpolating or extrapolating the behavior of fluid animations.

Sato et al. [**101**] propose to synthesize a velocity field by combining precomputed flow fields. First, a user selects a portion Ω_1 of a simulation with an associated velocity field $\mathbf{u}_1$. The user then pastes this region into another simulation Ω_2 with an associated velocity field $\mathbf{u}_2$. A margin Ω is created around Ω_1 within Ω_2, with a width specified by the user. The system then generates an interpolated velocity field $\mathbf{u}_\Omega$ in the margin Ω. This process is done by minimizing an objective function consisting of two terms: the sum of the velocity gradients across the boundary regions and the norm of a divergence of velocities. The following minimization problem is solved:

$$\mathbf{u}_\Omega = \arg\min_{\mathbf{u}} \sum_{\Omega} \left(||\nabla \mathbf{u}||^2 + \alpha \, ||\nabla \cdot \mathbf{u}||^2 \right), \tag{2.4.1}$$

with the boundary conditions $\mathbf{u}_\Omega = \mathbf{u}_1$ at $\partial\Omega_1$, and $\mathbf{u}_\Omega = \mathbf{u}_2$ at $\partial\Omega_2$. A numerical solution of this problem is obtained by taking the derivative of the above expression before using the biconjugate gradient stabilized method. Sato et al. [**102**] propose a few improvements over

Fig. 2.4.1. Combining simulations using the method of Sato et al. [**102**]. Left: Portions of two smoke simulations are combined to form a sandstorm. Right: The resulting animation.

their method. First, an optional frame matching procedure is introduced. The goal of this procedure is to automatically find the temporal offset to apply on the pasted animation, resulting in a maximal correlation between the density distributions of both animations in region Ω_1. The objective function to minimize is also replaced by:

$$\mathbf{u}_\Omega = \arg\min_{\mathbf{u}} \sum_{\Omega} \left(||\nabla \times \mathbf{u} - \nabla \times \hat{\mathbf{u}}||^2 + ||\nabla \cdot \mathbf{u}||^2 \right), \tag{2.4.2}$$

with $\hat{\mathbf{u}}$ the velocity field obtained by advecting the velocity field corresponding to the state of the fluid at the last frame, $\nabla\times$ the curl operator, and with the boundary conditions $\mathbf{u}_\Omega = \mathbf{u}_1$

at $\partial\Omega_1$, and $\mathbf{u}_\Omega = \mathbf{u}_2$ at $\partial\Omega_2$. Enforcing a similarity between the velocity field we are solving for and the velocity field $\hat{\mathbf{u}}$ allows us to take into account the momentum equation and to avoid too smooth flows obtained by simply minimizing the magnitude of $\nabla \times \mathbf{u}$. Results obtained using the proposed method can be found in Figure 2.4.1.

Assuming that vertical slices of a 3D breaking wave mimics the dynamics of a 2D wave, Mihalef et al. [**73**] introduce the Slice Method framework that generates controllable 3D breaking waves by defining 2D shape profiles at a given time t using a library of breaking waves. The library is built by generating initial conditions and by simulating the corresponding liquids using a 2D solver in order to capture the breaking of each wave. When

Fig. 2.4.2. Slice Method framework from Mihalef et al. [**73**], used to generate a breaking wave animation from 2D shape profiles.

capturing these data, the camera used is transposed along at the wave speed in order to keep the slices aligned. Geometric information as well as velocity field are stored and can then be linearly interpolated to obtain 3D data used to generate the 3D simulation. The animation before t can be generated by flowing the liquid backward in time using the precomputed 2D dynamics, while the animation after t is simulated forward in time using a standard 3D solver. Figure 2.4.2 shows a breaking wave generated using the method of Mihalef et al. [**73**] at five different times. Note that Brousset et al. [**17**] propose another approach to generate controllable breaking waves by using wave forces, with parameters explicitly affecting the characteristics of the wave.

Raveendran et al. [**97**] present a method to blend two existing liquid animations (A and B) with changing topologies. Each liquid animation is represented as a sequence of triangulated meshes describing the liquid surface and including per-vertex velocity data. A 4D surface model is created by generating a vertex pairing between consecutive frames. Using this representation, a nonrigid closest point algorithm is used to connect the two input surfaces. The main idea here is to deform mesh A (in both space and time) using a small number of local deformation nodes in order for each vertex from A to end up at the nearest point to it on surface B. In order to help the algorithm find good corresponding points, the user can provide a sparse set of correspondences between pairs of points on A and B. Once the closest

point on B is found for each vertex of A, a set of vertices describing the interpolated surface can be obtained by interpolating the vertices from A and their corresponding points on B. In order to obtain the final mesh describing the interpolated surface, a tetrahedralization of the entire space-time mesh is constructed and sliced with hyperplanes of constant time t. Thuerey [**122**] improves on this strategy by fully automating the matching process. Using grid-based 4D signed-distance functions to describe two input simulations A and B, a dense space-time deformation is calculated by using optical flow. Instead of using a single optical flow solve, Thuerey proposes to perform residual optical flow iterations until convergence (i.e., until A, once deformed, matches B). The sequence of deformations are then combined into a deformation field Φ using an alignment technique introduced in the paper. Assuming that A deformed using Φ is now sufficiently close to B, the author proposes to perform a final correction on Φ by projecting the deformed surface of A on B. The deformation obtained by marching along the negative gradient of the SDF is used to correct Φ. Once found, the deformation field can be used to efficiently interpolate the input simulations. Using signed-distance functions to represent the fluid allows the author to perform robust interpolation between fluids with different physics, for example between smoke and liquid.

Interpolating fluid simulations has several advantages. Instead of systematically resimulating fluid animations, relying on interpolation can be a solution to favor the reuse of existing simulation data, therefore saving the cost of simulation in some cases. This type of strategy can also lead to intuitive systems. Once the registrations of the phenomena to interpolate are performed, the techniques proposed by Raveendran et al. [**97**] and Thuerey [**122**] allow a user to efficiently explore the space of interpolated animations. In practice, interpolation seems to be restricted to input simulations with similar behaviors, and might produce unrealistic results in other cases because the interpolated animation is not physically correct.

2.5. Simulation Upresolution and Enrichment

In a production context, artists often rely on an iterative creation process to generate a target animation. After applying a control on the simulation, a new simulation is generated. Performing control directly on a high-resolution model leads to long iterations of the creation process, therefore hampering artistic creativity. The naive solution consisting in controlling a low-resolution simulation before rerunning the simulation in a finer grid usually produces a simulation exhibiting different behaviors. It can be explained by several factors, including the change of artificial viscosity, or the variation of results obtained after performing a pressure projection. Upresolution and enrichment techniques allow a user to control a low-resolution simulation before generating a higher-resolution version of the art-directed simulation, enhanced with fine details. Different strategies have been proposed to set up such a coarse-to-fine design cycle.

2.5.1. Tracking/Guiding

Some methods propose to simulate a high-resolution model while enforcing a proximity with a low-resolution input one. This strategy can be refered as *tracking* or *guiding*.

Following this idea, TRACKS [**12**] has been introduced to simulate a high-resolution version of a thin shell simulation, while keeping it close to a low-resolution guide animated model. In this method, low- and high-resolution discretizations of a same surface are decomposed into patches with a pairing between low- and high-resolution patches. TRACKS

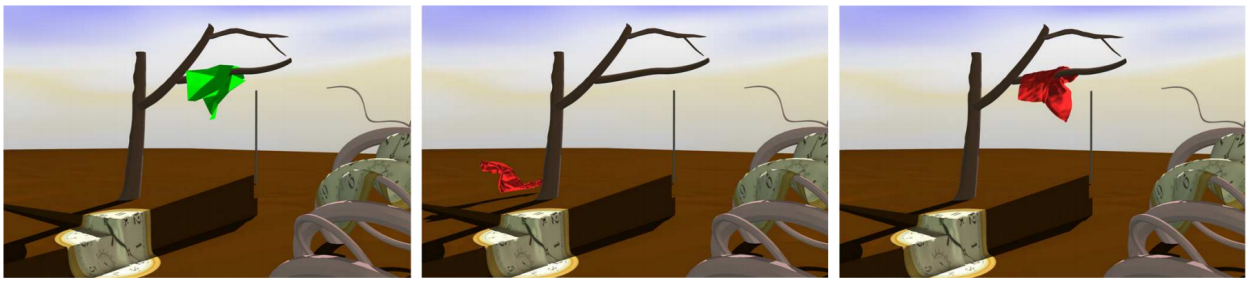


Fig. 2.5.1. Example of tracking using the TRACKS [**12**] framework. Left: Low-resolution guide simulation. Center: High-resolution free simulation, generated with the same initial conditions than the guide simulation. Note that the simulation does not exhibit the same behavior than the guide. Right: High-resolution tracked simulation. The high-resolution simulated piece of cloth closely follows the guide, but is enhanced with small-scale physically simulated details.

constrains the high-resolution surface (called *tracked* shape) of the object regarding to its low-resolution animated version. More precisely and for each patch of the high-resolution

surface, a constraint is added in order for its generalized center of mass to match the center of mass of its low-resolution counterpart. As a result, the high-resolution surface closely follows the behavior of the low-resolution one while being enhanced with physically simulated details as illustrated in Figure 2.5.1. Note that an application of this method has been proposed for directable hair simulation [**74**].

Methods generating a high-resolution fluid simulation by tracking the behavior of a low-resolution one have also been proposed. Nielsen et al. [**83**] modify the pressure projection step of a classical grid-based smoke simulation in order to constrain the low-frequency component of the velocity field to match the velocity field of a low-resolution input guide simulation. The traditional pressure projection step of a velocity field $\tilde{\mathbf{u}}$ can be formulated as the following saddle point problem:

$$\int_{\Omega} \left(\frac{1}{2} \left| \mathbf{u}(\mathbf{x}) - \tilde{\mathbf{u}}(\mathbf{x}) \right|^2 - \lambda(\mathbf{x}) \nabla \cdot \mathbf{u}(\mathbf{x}) \right) d\mathbf{x}, \tag{2.5.1}$$

where $\lambda(\mathbf{x})$ are Lagrange multipliers corresponding to pressure values satisfying the $\nabla \cdot \mathbf{u}(\mathbf{x}) = 0$ constraint, $\tilde{\mathbf{u}}(\mathbf{x})$ is the velocity field we want to project, and $\mathbf{u}$ is the velocity field we are solving for. Nielsen et al. [**83**] modify this formulation by inserting a tracking

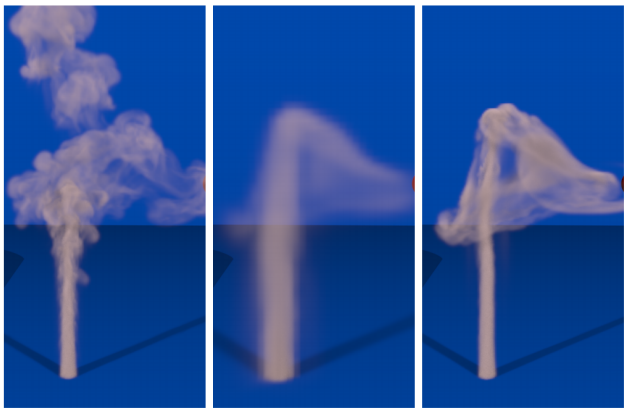


Fig. 2.5.2. High-resolution 3D smoke simulation generated by tracking a low-resolution guide using the method of Nielsen et al. [**83**]. Center: A low-resolution simulation ($16 \times 64 \times 16$). Left: A high-resolution simulation ($64 \times 256 \times 64$) generated using the same initial conditions than the low-resolution one. Right: A high-resolution simulation ($64 \times 256 \times 64$) generated by tracking the low-resolution simulation. Although the coarse aspect of the guide is transcribed, the distribution of smoke density is not faithfully reproduced.

component prescribing that a low-pass filtered version of the simulated velocity field should be as close as possible to the low-resolution guide velocity field up-sampled to

high-resolution. The resulting saddle point problem is given by:

$$\int_{\Omega} \left(\frac{\alpha(\mathbf{x},t)}{2} \left|\mathbf{u}(\mathbf{x}) - \tilde{\mathbf{u}}(\mathbf{x})\right|^2 - \lambda(\mathbf{x}) \nabla \cdot \mathbf{u}(\mathbf{x}) + \frac{(1-\alpha(\mathbf{x},t)}{2} \left|\left[F * \mathbf{u}\right](\mathbf{x}) - \mathbf{u}_{low}(\mathbf{x})\right|^2 \right) d\mathbf{x}, \tag{2.5.2}$$

with $\alpha(\mathbf{x},t)$ a scaling parameter. Note that the low-frequency component of the solution is a blend of the low frequencies in $\tilde{\mathbf{u}}(\mathbf{x})$ and $\mathbf{u}_{low}(\mathbf{x})$. In order to solve for the stationary point of this saddle point problem, it is reformulated as a linear system of $(D+1)N$ equations with $(D+1)N$ unknowns, with D the dimension and N the number of grid points. The multigrid method is employed to solve this system, returning a vector satisfying the previous equation and containing the D velocity components as well as the Lagrange multiplier for each grid cell. To handle boundary conditions, Nielsen et al. [**83**] use the penalization method. The main idea is that the system is not explicitly aware of boundaries in the scene and solves for velocity and pressure values everywhere. However, inside the boundaries, the velocity field is penalized towards the velocity of the boundary. The resulting method produces good results at a reasonable cost, but has several limitations. As highlighted by the authors, the visual impact of density-diffusion and density-dissipation depends on the resolution of the simulation. For this reason, smoke density is not always well reproduced, as shown in Figure 2.5.2. Also note that when using a time-dependent $\alpha(x,t)$, the matrix used in the linear system needs to be recomputed at each iteration. Nielsen and Christensen [**82**] tackle this last issue by making the left-hand side of the linear system independent of the guiding weights $\alpha(\mathbf{x},t)$.

Using ADMM, Gregson et al. [**39**] simulate a high-resolution fluid by tracking low-resolution tomographic scannings using optical flow. The method solves for physically plausible dense velocity fields. In order to design the method, the authors start from the observation that the pressure solve can be interpreted as a proximal operator. Using this idea, the following minimization problem is formulated:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} E_{PC}(\mathbf{u}) + \alpha E_{SM}(\mathbf{u}) + \beta E_{KE}(\mathbf{u}) + \gamma E_{DIV}(\mathbf{u}), \tag{2.5.3}$$

where E_{PC} is the optical-flow photoconsistency, E_{SM} is a smoothness term, E_{KE} is a kinetic energy penalty, and E_{DIV} is a divergence-free term, as in the previous section. Two functions $F(\mathbf{u}) = E_{PC}(\mathbf{u}) + \alpha E_{SM}(\mathbf{u}) + \beta E_{KE}(\mathbf{u})$ and $G(\mathbf{u}) = E_{DIV}(\mathbf{u})$ are defined as well as their corresponding proximal operators. Minimizing F leads to good fluid tracking, while minimizing G leads to a divergence free solution.

Huang et al. [**46**] propose to generate match points in the simulation domains of both the low-resolution guide and the high-resolution smoke, at various fixed positions in space

with Gaussian kernels of fixed radius. The position and radius of each match point are either defined by a user or by automatically sampling the domain. Using these points, a matching procedure is applied on the high-resolution simulation. Scalar or vector physical quantities that should be tracked (e.g., density, temperature, velocity) are sampled in both simulation domains. A weighted error is then computed and used to apply a correction force proportional to it. This force is then applied to the high-resolution simulation. Results obtained using this strategy are shown in Figure 2.5.3.

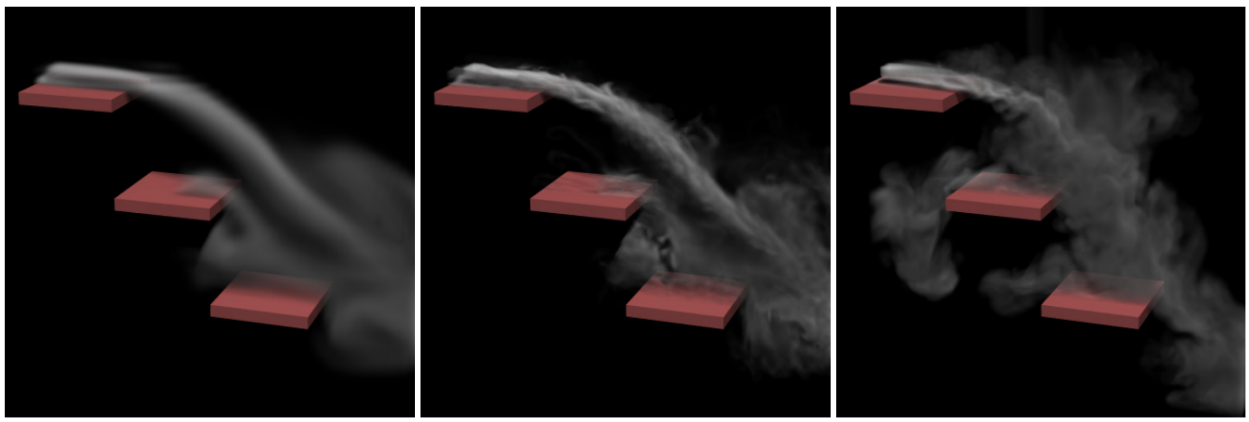


Fig. 2.5.3. High-resolution smoke guiding from Huang et al. [**46**]. Left: A coarse smoke simulation. Center: A high-resolution simulation guided using the method of Huang et al. [**46**]. Right: A high-resolution free simulation.

At each time step of a high-resolution smoke simulation, Forootaninia and Narain [**33**] extract the low-frequency component of the velocity field using a low-pass filter in the frequency domain, and constrain it to match the corresponding frequencies of an input low-resolution guiding velocity field. The initial high-frequency component is then combined with it, resulting in a high-resolution guided simulation. Note that the amount of guiding can be tweaked by adjusting the filtering cutoff.

Kim et al. [**59**] enhance the visual appearance of a level-set-based simulation by performing a wave simulation on the surface of a low-resolution liquid using the Closest Point Method (CPM). This method can be applied to a particle-based fluid by transferring particle data to a grid. Mercier et al. [**72**] tackle this issue by adding turbulent details directly at the particle level. First, a set of input particles from a low-resolution simulation is converted into a high-density surface point set. This high-density set of particles is then used to perform a wave simulation, robustly generating fine and temporally coherent details to the liquid surface.

Yuan et al. [**137**] use flow patterns extracted from a coarse smoke simulation to guide a final high-resolution simulation. The Lagrangian Coherent Structure (LCS), a characteristic of a dynamical system used in the field of physics and meteorology to study real-world fluid dynamics, is extracted using the finite-time Lyapunov exponents (FTLE), a method that measures the rate of separation of neighboring chunks of fluid after a time interval. The LCS, obtained by computing the local maximal of the FTLE field, is processed by applying a thinning method in order to extract a volumetric skeleton. The resulting volume becomes the control domain, a region where the guiding of the final high-resolution simulation occurs. The high-resolution fluid guiding is performed by applying a guiding force field that minimizes the difference between the simulated high-resolution velocity field and the low-resolution velocity field up-sampled at high resolution.

Recently, Sato et al. [**99**] propose to guide a fluid by solving a minimization problem in a stream function space. For each time step t, the input low-resolution flow is first up-sampled at high resolution, and converted to a stream function $\Psi_t(\mathbf{x})$. A guided stream function is then defined as $\hat{\Psi}_t(\mathbf{x}) = \Psi_t(\mathbf{x}) + w_t(\mathbf{x})\Psi_{usr,t}(\mathbf{x})$, where $w_t(\mathbf{x})$ is a scaling function, and $\Psi_{usr,t}(\mathbf{x})$ is a user-defined stream function. The following minimization problem is defined:

$$w_t^* = \underset{w_t(\mathbf{x})}{\arg\min} \sum_{\mathbf{x}} \left|\left|\nabla \times \hat{\Psi}_t(\mathbf{x}) - \mathbf{V}_t(\mathbf{x})\right|\right|^2 + \alpha(w_t(\mathbf{x}))^2, \tag{2.5.4}$$

with $\mathbf{V}_t(\mathbf{x})$ obtained by advecting the velocity field at the last frame. Note that solving for scaling function $w_t(\mathbf{x})$ reduces by three the number of parameters to solve for. Also note that no projection is required, since the curl of a potential field returns a solenoidal vector field. In the objective function, the user-defined parameter α can be tweaked to adjust the strength of the regularization term and, therefore, to adjust the strength of the control applied to the simulation. After taking the derivative of the objective function with

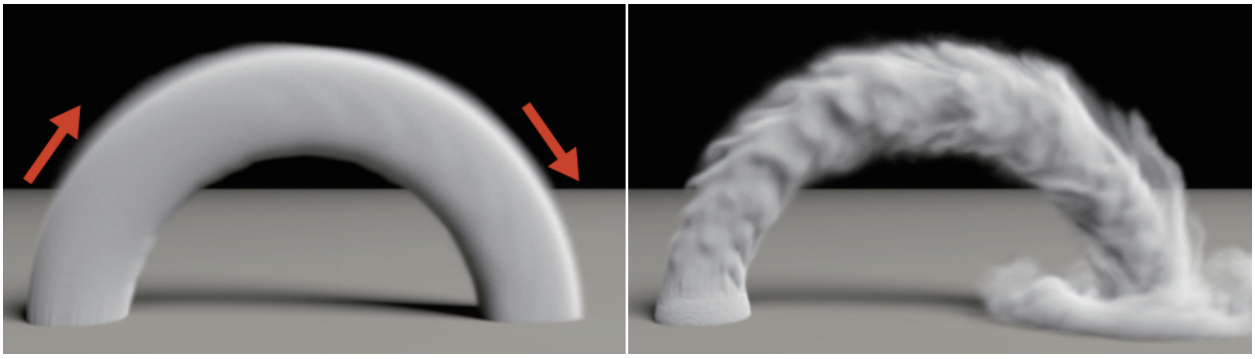


Fig. 2.5.4. Left: Density field advected by a procedurally-generated low-resolution flow field. Right: High-resolution simulation generated using the method of Sato et al. [**99**].

respect to $w_t(\mathbf{x})$ and solving the minimization problem using the conjugate gradient method, the final high-resolution guided velocity field can be obtained by computing the curl of the guided stream function $\hat{\Psi}_t$. Note that this method can be used to guide a smoke simulation using a procedural flow field as shown in Figure 2.5.4.

Tracking/guiding techniques perform a simulation constrained by a low-resolution model. Since they are simulated, high-frequency details are usually more plausible than if they were generated using a procedural method. These techniques come at least with the cost of simulating a high-resolution simulation, which is usually not a problem since the upresolution stage can be done only once after producing the coarse model, and usually does not require as much input from an artist.

2.5.2. Procedural Methods

Instead of simulating high-frequency details, some methods propose to generate them using procedural techniques.

Using a wavelet representation, Kim et al. [**60**] enhance the velocity field of an input low-resolution smoke simulation with procedural details during a post-processing step. To do so, the velocity field of the input simulation is first up-sampled at the high resolution. This operation does not generate any new eddies in the new spectral bands. Kolmogorov's theory, which gives the energy spectrum of a turbulent fluid, is then used to estimate how much energy should be injected in the new spectral bands. A procedural incompressible turbulence function is then used to re-synthesize missing high-frequency details. In order to detect forward scattering (i.e., an eddy breaking into two eddies of half its size, usually happening when an eddy is streched/compressed during advection), texture coordinates are advected along with the flow. The amount of local deformations is quantified by computing the Jacobian of the texture coordinates, and used to trigger turbulence regeneration if an eddy has forward scattered into a higher spectral band. This method is efficient, and guarantees that the new high-frequency details do not interfere with existing lower-resolution frequencies by only generating details over a narrow spectral band. A high-resolution simulation generated by increasing by a factor of 256 the resolution of an input simulation using the method of Kim et al. [**60**] is shown in Figure 2.5.5.

Zhao et al. [**142**] enhance the velocity field of a low-resolution fluid simulation using a sequence of divergence-free noise vector fields in the spectral domain. The characteristics of the turbulence vector field are controllable, allowing an animator to add details in a user-defined spectral band of the velocity field. The resulting vector fields are combined and

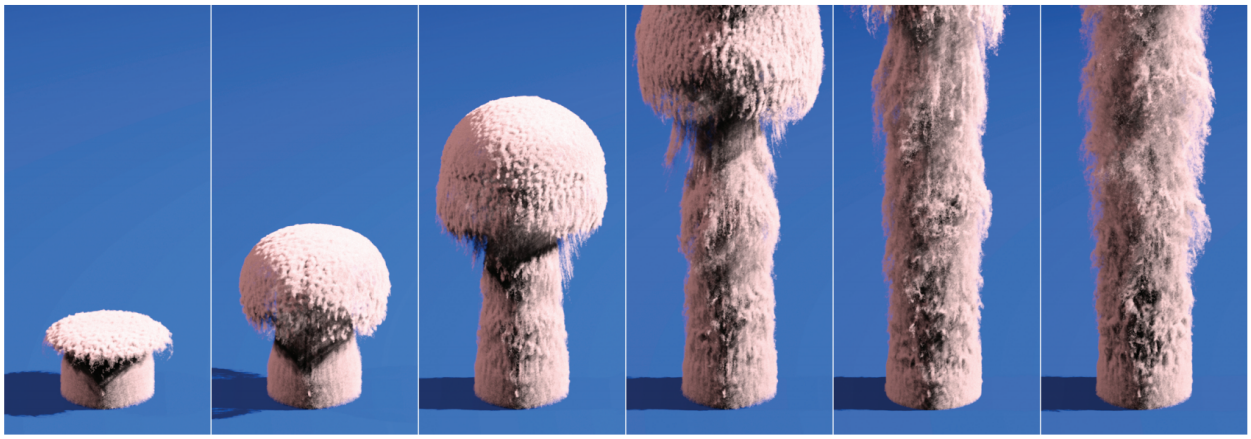


Fig. 2.5.5. A $12800 \times 25600 \times 12800$ generated by adding eight turbulence spectral bands to an input $50 \times 100 \times 50$ smoke simulation using the method of Kim et al. [**60**].

used as a turbulence force field to agitate the simulation flow. The sequence of synthetic force fields are precomputed and applied to the system at run-time. Note that a turbulence enforcement can be applied in local areas, in particular time ranges. Narain et al. [**79**] use incompressible turbulent velocity fields generated by a turbulence model to create subgrid-scale flow details in a classical fluid solver. Pfaff et al. [**91**] simulate the coarse aspect of a

Fig. 2.5.6. Low-resolution smoke simulation enhanced with turbulent details using the method of Pfaff et al. [**91**]. From left to right: Low-resolution smoke simulation ($32 \times 8 \times 32$), and corresponding high-resolution simulations generated using 250k, 1M, and 4M turbulence particles.

smoke simulation using a fluid solver, but rely on a system of anisotropic turbulence particles to generate fine details of turbulent flows on it. Note that particles are not interacting with each other, making the method trivial to parallelize. The turbulence field is described by its statistical properties using a modified $k - \epsilon$ turbulence model, and evaluated only where needed. For each particle, a velocity is computed by summing the interpolated low-resolution velocity field and both an isotropic and an anisostropic turbulence velocity based on curl noise. Results obtained using this technique are shown in Figure 2.5.6.

Compared to simulated high-frequency details, procedural details appear usually less plausible, especially for large upresolution factors. Since solenoidal procedural vector fields are often used, the augmented velocity field satisfies the conservation of mass but does not satisfy conservation of momentum. These methods are usually very efficient, and offer great results when used with a relatively small upresolution factor, to add small-scale details during a final beautification pass, such as done by Gregson et al. [**39**].

2.5.3. Enforcement of Boundary Conditions

Nielsen and Bridson [**81**] present a method to constrain a high-resolution liquid simulation to stay in a thin outer shell around the surface of a low-resolution guide liquid. First, a guide is built by simplifying the low-resolution guide using several operations including morphological operators such as erosion and dilatation. A high-resolution liquid is then simulated in a band of user-specified thickness, using custom velocity boundary conditions at the interface between the guide and the guided volumes. These velocities are obtained by interpolating the velocity field of the low-resolution simulation in a layer surrounding the surface, and are used as Neumann boundary conditions during the pressure projection step of the high-resolution simulation. Note that liquid can be reseeded or removed at each time

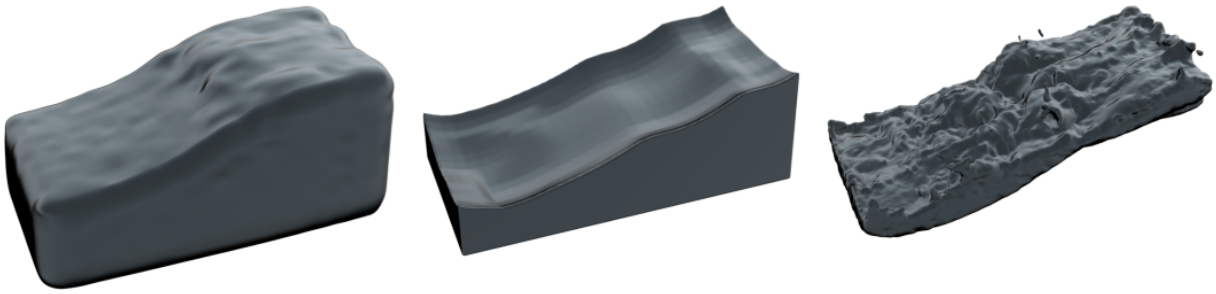


Fig. 2.5.7. High-resolution liquid guided using the method of Nielsen and Bridson [**81**]. From left to right: Low-resolution input liquid, guide computed from the input liquid, and high-resolution liquid simulated in a thin shell around the guide.

step to prevent liquid to leak or air regions to appear at the interface with the guide shape. Since the solve is performed only in a small region of space, the method is usually faster than performing a full unconstrained simulation. This strategy is illustrated in Figure 2.5.7. Note that this technique has inspired Stomakhin and Selle [**118**], and has been integrated into Autodesk's Bifrost [**85**].

While techniques such as those presented by Kim et al. [**59**] or Mercier et al. [**72**] perform a wave simulation on a liquid surface, Nielsen and Bridson [**81**] propose to simulate a volume of high-resolution liquid in a band on top of a low-resolution liquid. This technique is an elegant alternative to obtain a plausible but less constrained high-resolution liquid simulation, and should generate plausible results when used with large upresolution factors.

2.5.4. Data-Driven and Learning-Based Techniques

Over the past few years, machine learning has been used as a mean of up-sampling a coarse flow simulation to obtain finer and more visually-pleasing results inferred from a training set of fluid data.

Sato et al. [**104**] use high-resolution precomputed velocity fields simulated in 2D to increase the resolution of a 3D low-resolution fire simulation at run-time. When generating the 2D simulations, the velocity fields at each time step are subdivided into small blocks. For each of these blocks, a principal component analysis is executed, and the resulting principal components are stored in a database. To synthesize a high-resolution 3D fire, the principal components stored in the database are downsampled to the resolution of a low-resolution 3D simulation, before computing a weighted sum of these fields locally minimizing the differences with the velocity stored in the current 2D layer in a least-squares sense. Once found,

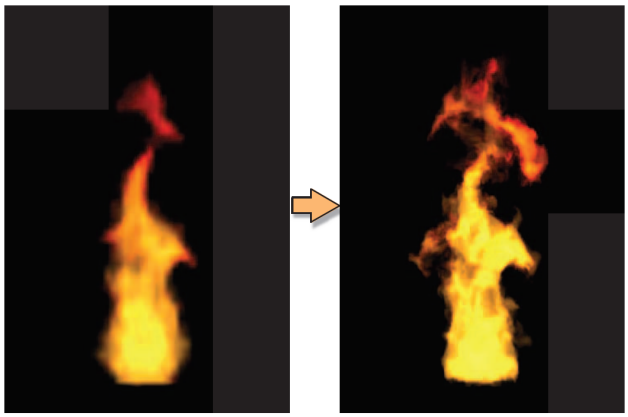


Fig. 2.5.8. Detailed fire produced from a coarse input using the method of Sato et al. [**104**].

the coefficients can be used to combine the original blocks of velocity in order to synthesize a high-resolution flow field. Note that this process is performed for each dimension of the 3D simulation, using the horizontal component in 2D (i.e., X) for the horizontal components in 3D (i.e., X and Z), and the vertical component in 2D (i.e., Y) for the vertical component

in 3D (i.e., Y). For high-resolution ratios, the method is applied recursively, increasing the resolution gradually. Results obtained using this technique can be found in Figure 2.5.8.

Chu and Thuerey [**23**] synthesize high-resolution smoke animations from space-time flow data. Convolutional neural networks (CNNs) are used to generate descriptors for smoke density and velocity. Simulations with randomized initial conditions are used to generate the training data. For each set of initial conditions, a high- and a low-resolution simulation are generated. After a user-defined duration and in order to keep both simulations close to each other, the coarse simulation is re-initialized by low-pass filtering the high-resolution simulation. Small deforming patches are seeded throughout the volume, and used to robustly track the behavior of the fluid over time in order to record the full coarse and fine velocity and density data to populate a data collection. A deformation-limiting strategy is also used to make sure that each patch does not become ill-shaped over time when advected. Note that the training is performed on similar and dissimilar pairs to improve the efficiency of the algorithm. The run-time high-resolution flow synthesis is then divided into two passes: a first forward pass during which new patches are seeded, advected and faded out, and a second backward pass in which newly created patches are advected backwards in time over the course of the fade-in interval. At render time, the low-resolution simulation density as well as the information related to each patch are used to reproduce a high-resolution version of the coarse simulation, as shown in Figure 2.5.9.

Fig. 2.5.9. Coarse smoke simulation and corresponding high-resolution smoke synthesized using the method of Chu and Thuerey [**23**] at three different times.

Xie et al. [**133**] share the same goal but use a conditional generative adversarial network (GAN) with the velocity of the fluid flow as a conditional input. This strategy allows the network to learn the underlying physics and improves the temporal coherency of the generated features. Training a neural network with a large number of weights can be expensive and challenging. To circumvent this problem, Werhahn et al. [**128**] introduce multi-pass GAN, which reduces the dimensionality of the problem by breaking down the generative

task into multiple orthogonal passes, each of them solving a more manageable problem. In this method, a 3D volume of fluid data is considered as a stack of 2D slices used for training. Processing the data multiple times on orthogonal slices makes the multi-pass GAN able to infer a full 3D motion. Note that recently, a method using a conditional GAN [**134**] has also been introduced to generate a static volumetric splash model from a sketch. Um et al. [**127**] introduce a data driven approach to enhance liquid simulations with splashes using neural networks.

Xiao et al. [**132**] use a learning-based flow correction to fast-preview a high-resolution version of a coarse smoke simulation. Sharing the same goal, Bai et al. [**5**] use dictionary learning to perform smoke upresolution. For each frame of an input coarse smoke animation, the method seeks a sparse representation of small, local velocity patches of the flow based on an over-complete dictionary, and uses the resulting sparse coefficients to generate a high-resolution animation. The neural network used learns both an evaluation of sparse patch encoding, and a dictionary of corresponding coarse and fine patches.

Fig. 2.5.10. Stylized smoke simulations generated using TNST [**56**]. Each of the six images depicts a stylized smoke density field as well as the 2D texture used to generate it.

Sato et al. [**100**] draw from style-transfer methods from image editing to transfer the turbulence style from a simulation to another one. Specifically, example-based image synthesis techniques are extended to handle velocity fields using a combination of patch-based and optimization-based texture synthesis. Neural Style Transfer (NST) refers to another popular class of algorithms that have been used in the field of image and video processing to transfer the appearance of a media to another media. Inspired by this strategy, Kim et al. [**56**] introduce the Transport-based Neural Style Transfer (TNST) method that transfers styles and semantic structures from a natural image to a grid-based smoke simulation in order to generate art-directed smoke animations as shown in Figure 2.5.10. By rendering the smoke from different viewpoints using an end-to-end differentiable renderer, Kim et al. [**56**] solve for a velocity field that leads to an advected 3D smoke density showing features of the input 2D texture from the different viewpoints we define. To generate such a velocity field, a Convolutional Neural Network trained for natural image classification is used, making it

possible to transfer low (i.e., edge, patterns) to high (i.e., complex structures) level features from the texture to the smoke. Note that the velocity fields are computed independently at each time step. In order to improve the temporal coherency of the generated velocity field, stylized velocity fields are aligned for a given window size, which can be expensive for large window sizes. Kim et al. [**57**] tackle this issue by extending TNST to fluids simulated in a Lagrangian setting. The resulting method optimizes for per-particle attributes, ensuring a better temporal consistency of the solution at a reduced cost.

Results obtained by doing upresolution using learning-based techniques are undoubtedly very promising. These techniques are more and more physics inspired, leading to more plausible and temporally coherent results. These methods are not without downsides. For example, one could note that the quality of the generated results highly depends on the training set used. Changing the simulation parameters used to generate the training set requires to retrain the model, which can be very time consuming since a large amount of data and long training times are still required.

2.6. Conclusion

Various methods have been proposed to control a fluid and increase the resolution of its simulation. Some of these methods can already be combined to implement the two-stage fluid simulation control process we described in Chapter 1. Compared to techniques that are directly affecting the velocity field, strategies relying on the use of control forces are of interest because they conserve the momentum of the fluid and result in more plausible dynamics. However, reaching a specific target using a handmade force field is a tedious task requiring a long trial-and-error process when applicable. Ideally, a user should be able to simply express a target to reach, letting the system generate the control required to match it, or indicate how and where it cannot satisfy his intentions. Although various strategies have been proposed, many things still have to be done in the field of fluid control. In particular, effort should be made to provide new intuitive and user-friendly methods, allowing users to quickly obtain a targeted behavior. Upresolution techniques can also be improved. While data-driven and machine-learning techniques offer very promising results, we do believe that simulation-based techniques are still of interest when the objective is to generate plausible fine details that are temporally coherent.

Chapter 3

Particle-Based Liquid Control using Animation Templates

3.1. Introduction

Controlling a fluid simulation is a daunting, but necessary task. A simulation depends on many parameters, including its initial state, its time integration scheme, and the underlying physical model used to update the physical state. Modifying the initial state may drastically change the behavior of a simulated fluid, most of the time in an unpredictable manner. In addition to that, the parameter space offered by this strategy is too limited to cover a wide range of target animations (especially nonphysical ones). For these reasons, specialized control methods modifying the behavior of a simulation in a more predictable manner should be developed and used when art-directing a fluid. As described in Chapter 1, art-direction could take the form of a two-stage process: a first interactive editing stage, and a second upresolution stage. In this chapter, we describe a liquid control system that could be used to implement the first stage of such a pipeline.

Fluid simulation control in computer graphics has been investigated for more than two decades, but we believe that research is still needed to provide user-friendly tools allowing animators to generate art-directed simulations. In particular, such methods should be efficient, in the sense that they should not add a big computational overhead on top of an uncontrolled simulation, allowing an interactive editing stage when possible not to hamper the creative process. In addition to that, a control method should be intuitive. While this characteristic is very subjective, we consider that an intuitive method should be easy to understand and to specify the inputs required to generate a targeted simulation.

This chapter introduces a particle-based liquid simulation and editing system that relies on the use of precomputed simulation templates stored in a database. Using our system, a user can select a precomputed template describing an animation (e.g., a wave, drop of water, etc.), visualize it, and use it to control a liquid simulation. Our system allows a

user to compose a target animation using several templates created at different locations and times. The system then performs a full resimulation of the liquid animation, using the set of templates created by the user to locally control the liquid and reproduce the created templates. Specifically, each template instance contains a set of animated control particles. When resimulating the animation, each control particle is used to generate a control force field applied to the neighboring liquid particles. Attraction, repulsion, and velocity forces are used to locally control the behavior of the liquid, reproducing the target liquid distribution and dynamics from the template. Temporary particles are created and removed when no longer needed. During their life span, these special particles behave like standard liquid particles but they help to efficiently reproduce the animator's vision with fewer requirements on the initial state and on the surrounding environment of the simulation. The resulting animations are both plausible and predictable. In addition to that, the design choices we made lead to a relatively small computational overhead compared to an uncontrolled simulation. In our system, each template instance can be modified using a set of traditional animation-editing metaphors. These metaphors, familiar to computer graphics artists, include spatial and temporal transformations. We found that allowing users to transform the templates improves their reusability. In order to help an animator compose a liquid animation using several templates, we built an intuitive graphical user interface inspired by classical 3D content production and video editing software. The resulting system is easy enough to be used by a nontechnical user.

The completion of this project resulted in a paper [**105**] published in the journal *Computer Graphics Forum*, presented at the *ACM SIGGRAPH / Eurographics Symposium on Computer Animation* 2020. An accompanying video can be found on the webpage (`http://arnaud-schoentgen.com/publication/liquidtemplate/`) dedicated to this project.

The rest of this chapter is organized as follows. We first discuss related work in Section 3.2. We describe our method in Section 3.3, and provide details of our implementation in Section 3.4. Results are then presented and analyzed in Section 3.5, before conclusions and future challenges are drawn in Section 3.6.

3.2. Related Work

Treuille et al. [**125**], McNamara et al. [**71**], and Pan et al. [**88**] propose to use a sparse set of keyframes to control a fluid. These keyframes are generated by an artist either as sketches or meshes. Note that they can optionally include target velocities. These methods

guide the fluid to match the targets at user-defined frames. However, designing a sequence of keyframes leading to a plausible fluid animation is another complex task on its own. This type of approach relies on space-time optimization. Although major improvements have been achieved over the years, the algorithmic aspect makes this type of approach computationally expensive. Therefore, such techniques do not seem to be suitable if we want to provide quick visual feedback to an animator during the editing stage. Pan et al. [**86**] propose to reduce the time complexity of these techniques by performing the optimization for a single frame, for a user-defined subset of particles. The control is then propagated to neighboring frames and regions of space. The consequence is that interactive liquid editing is possible with this system, but the resulting behavior is much more approximate.

Other methods, such as the one by Inglis et al. [**49**], propose to control a simulation using a target velocity field. Since the density or particle distribution are the data that we use to extract the surface to render, we found that describing a target animation using a similar description (density or particle distribution) should lead to more predictable results than using a target velocity, which basically describes the temporal variation of these fields.

A few techniques focus specifically on providing a user-friendly system to generate art-directed simulations. Among these techniques, systems using a database of animated features are particularly interesting. Using such techniques, an animator can pick elements from the database and combine them to create an art-directed animation. In the system introduced by Manteaux et al. [**70**], an artist can extract animated regions of interest from a mesh describing a liquid surface, before saving them in a database. These space-time features can then be loaded and pasted onto another liquid surface. The resulting tool seems to be handy and is very inspiring, but unfortunately, limited to copy and paste features from a mesh describing a liquid surface. The approach used is purely geometrical and might be too limited to generate complex fluid animations. As Manteaux et al. [**70**] themselves note, physical consistency is not checked and issues may appear when pasting a surface with multiple connected components (e.g., in a splash). The system presented by Mihalef et al. [**73**] is another inspiring example. It allows an animator to use 2D slices of breaking waves selected from a library to define profiles of a 3D breaking wave. The resulting system can be used to intuitively generate controllable simulations. Unfortunately, it is limited to generate simulations describing the breaking of a wave, and generalizing it to different situations is tricky. Close to the concept of recording a portion of fluid simulation, Wrenninge and Roble [**131**] present a fluid simulation package introducing the concept of a *Water Recorder*: a user-defined region of space in which a liquid will be flowing through and which can be used to write level set and velocity fields of the fluid in files.

Some methods propose to use control particles to guide a simulation. Thürey et al. [**124**] introduce a model of control particle that generate two types of forces. A first attraction force is used to attract the particles towards the center of control particles. A second velocity force is used to reduce the velocity difference between particles and neighboring control particles. The system that we describe in this chapter can be seen as an extension of this model.

Our goal is to use portions of precomputed particle-based liquids to locally control the behavior of a simulation. At first sight, the method described by Bojsen-Hansen and Wojtan [**13**] could seem to share the same goal than us. It proposes to resimulate a portion of an already simulated liquid, for example to locally add a splash, an obstacle, or to increase the resolution of a liquid. In their method, newly generated waves are damped in order to not propagate outside of the controlled region. It allows a user to localize the editing in space, ensuring a smooth transition between the simulation and the previously simulated background liquid. However, their technique is not designed to compose a complex liquid simulation with many parts interacting together. We consider that the problem tackled by Bojsen-Hansen and Wojtan [**13**] is orthogonal to ours.

Note that extensions of control strategies based on SDF representations of both the liquid and the target [**111, 112**] could be investigated to solve the problem that we are tackling as part of this project. Extensions of methods using custom boundary conditions [**81, 96, 118**] could also be of interest, although finding the appropriate region of space to enforce custom boundary conditions would not be trivial. Enforcing Neumann boundary conditions on the borders of a domain containing the feature to reproduce (e.g., axis-aligned bounding box or convex hull) would result in a natural aspect of the simulation, but likely not controlled enough to reproduce the template. Using a mesh describing the template surface could be an option but one should think about how to attract the liquid to fill the template at the beginning of the control period. Instead, we decided to rely on control particles [**95, 124, 69**] to drive our simulation. This natural choice allows us to keep the same type of representation for both the simulated liquid and the target.

3.3. Our Approach

We introduce a method to enable artists to control particle-based liquid simulations using precomputed portions of simulations called *templates*, stored in a database. In our system, a scene consists of SPH particles discretizing either a single-phase or a multi-phase liquid, interacting with rigid bodies. When designing a simulation, a user can pick one or several precomputed animations in the database, edit them using basic affine transformations such

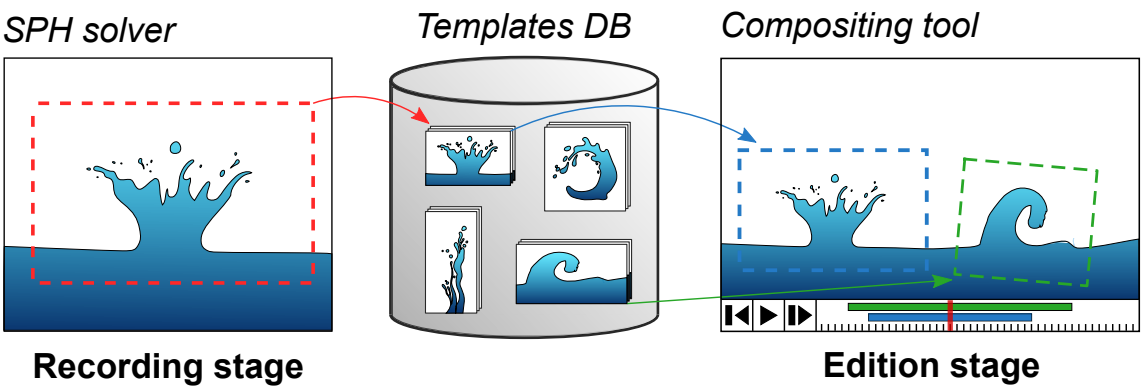


Fig. 3.3.1. Overview of our system. Particle-based animation templates are precomputed and stored in a database. The user selects a template and instantiates it with geometrical and temporal transformations. Our system uses all instances of templates to generate control forces in order to drive the current liquid to reproduce a targeted simulation.

as translations, rotations, scalings, or reflections, and instantiate them in the scene at given times. Animators can also perform temporal editing over each instance by cropping the beginning and/or the end of each instance to select a temporal region of interest, also changing its playing speed as well as its temporal direction (e.g., forward or backward). Animators can script the motion of each instance in the scene by defining different transformations at keyframes, the transformation affecting each instance at each time step being interpolated between keyframes. When editing a multi-phase liquid simulation, users must specify for each instance the phase to control. Once done, our system generates control forces and manages temporary particles to correctly reproduce the templates in both space and time. Figure 3.3.1 shows a schematic overview of our system, while the following sections provide details of its underlying concepts.

3.3.1. Liquid Simulation Template

Our liquid control system uses precomputed templates of liquid animations to generate control over a simulation. Each template corresponds to a liquid simulation of a given duration within a region of space; it is stored in a database. Each template stores the position, velocity, and mass of every particle in the recorded region of space at every frame for the duration of the animation. Global information such as the region size, the animation duration, the template header (e.g., name) are also stored. Templates can come from either realistic or non-realistic liquid animations. Note that no information about solids is stored.

Although we also use our system to record templates, any kind of particle-based liquid simulator could be used to record templates, as the information currently stored is standard.

Once recorded and stored as a binary file in the database, each template can be selected, loaded, and placed (i.e., instantiated) in space and time in a scene during the editing process. In order to expand artistic control over the animation, our system allows animators to apply spatial and temporal transformations to the instantiated templates. Once applied in scenes, templates are used to generate control in the areas covered by at least one template, called *controlled areas*. When a template is instantiated, its stored particles become control particles. These particles do not appear as liquid particles in the scene, but instead act as generators of control forces. Therefore, at each frame, the current distribution and properties of control particles, as well as the instance transformations, allow us to exert additional forces that are applied to the scene liquid particles located in every controlled area. Moreover, because not enough liquid particles may be present to reproduce motions in some regions, we introduce temporary liquid particles to efficiently minimize density differences between template and controlled areas.

In the rest of this chapter, simulated liquid particles (including temporary particles) are simply called *liquid particles*. Particles stored in an instantiated template are called *control particles*.

3.3.2. Control Forces

Each liquid particle located in a controlled area is subject to extra forces, called *control forces*, in order to drive the simulated liquid to locally reproduce the templated liquid behavior.

Attraction and Velocity Forces. In our system, each control particle generates in its neighborhood both an attraction force and a velocity force. These forces are inspired by the work of Thürey et al. [**124**]. Attraction forces aim to drive liquid particles into regions covered by control particles. Summing up attraction forces applied on a liquid particle located at position $\mathbf{x}$ yields

$$f_a(\mathbf{x}) = w_a \sum_j \alpha_j \frac{\mathbf{x}_j - \mathbf{x}}{d(\mathbf{x}, \mathbf{x}_j)} W\left(d(\mathbf{x}, \mathbf{x}_j), h\right), \tag{3.3.1}$$

with w_a a constant that defines the strength of the attraction force, $\mathbf{x}_j$ the position of the j-th control particle, and $d(\mathbf{x}, \mathbf{x}_j) = \|\mathbf{x} - \mathbf{x}_j\|$. The scaling factor α_j of the j-th control particle reduces the attraction force when control particle j is "covered" by enough liquid

particles. The scaling factor is defined as

$$\alpha_j = 1 - \min\left(\sum_i \frac{m_i}{\rho_i} W\left(d(\mathbf{x}_j, \mathbf{x}_i), h\right), 1\right), \tag{3.3.2}$$

with mass m_i, density ρ_i, and position $\mathbf{x}_i$ of the i-th neighboring liquid particle. As described by Thürey et al. [**124**], W is the *Poly6* kernel used by Müller et al. [**78**].

Since we want the liquid particles to locally exhibit the same kinematics as neighboring control particles, we apply a velocity force to reduce velocity differences between liquid particles and surrounding control particles. The velocity force applied on a particle located at position $\mathbf{x}$ moving with velocity $\mathbf{v}$ is given by

$$f_v(\mathbf{x},\mathbf{v}) = w_v \sum_j (1 - \alpha_j)(\mathbf{v}_j - \mathbf{v}) W\left(d(\mathbf{x}, \mathbf{x}_j), h\right), \tag{3.3.3}$$

with constant w_v defining the strength of the velocity force, and $\mathbf{v}_j$ the velocity of the j-th control particle. Note that, contrary to the original work from Thürey et al. [**124**], we multiply the influence of each control particle by its coverage when computing the velocity force. In practice, this coverage factor reduces the impact of velocity forces at the liquid interface, and lets attraction forces bring in more easily some liquid in regions covered by control particles. Both these complementary forces generated from control particles are depicted in Figure 3.3.2.

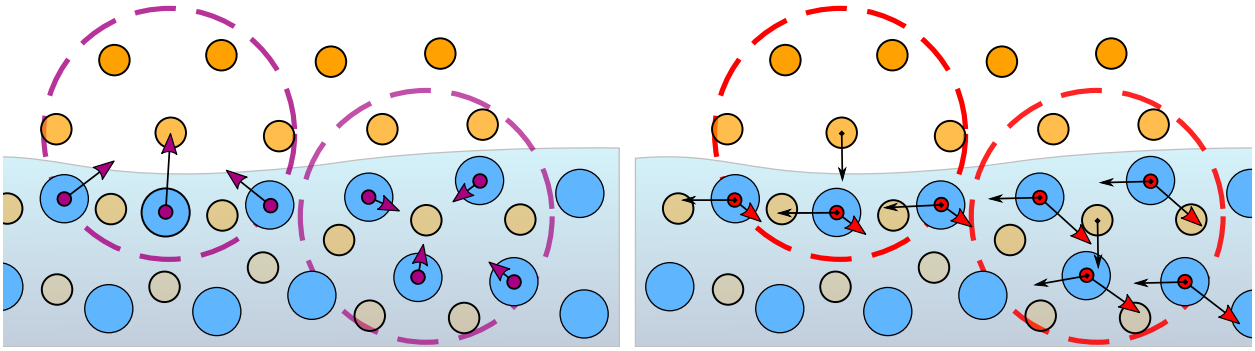


Fig. 3.3.2. Left: Attraction forces (purple arrows) generated for two different control particles, and applied on different liquid particles (in blue). Control particles are represented using different shades of orange: light orange for α close to 0, saturated orange for α close to 1. Attraction forces are maximal for control particles with little liquid coverage, and minimal for particles with a significant degree of coverage. Right: Velocity forces (red arrows) generated for the same control particles. Velocities of both control and liquid particles are represented with black arrows. Velocity forces are scaled down when emitted from a control particle with little coverage, since they could counteract the attraction forces.

Repulsion Layer. Control particles generate both attraction and velocity forces in their neighborhoods in order to attract and drive the controlled liquid to match the behavior of the template. However, we can observe surface dissimilarities between template and simulated liquids in many cases. Indeed, liquid particles that are not covered by control particles are unaffected by them, which means that they can move freely. This lack of control at the template surface causes difficulties to correctly reproduce that surface. To circumvent this limitation, we introduce a repulsion particle layer model illustrated in Figure 3.3.3.

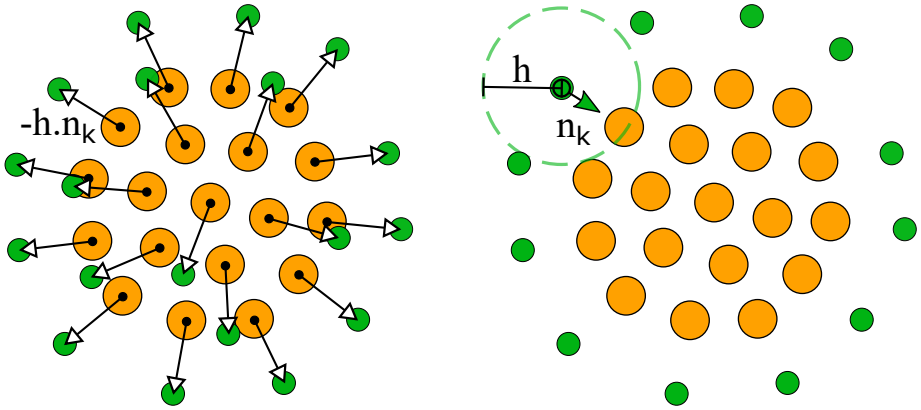


Fig. 3.3.3. Repulsion particle layer providing control at the template liquid surface. Left: Seeding repulsion particle candidates (green) using each control particle (orange). Repulsion particle candidates are generated at a distance h along the normalized negative density gradient. Right: Repulsion particles too close to a control particle are discarded. The remaining particles are stored and used to generate repulsion forces collinear to $\mathbf{n}_k$ in their neighborhood.

This repulsion layer is generated for the whole template duration during a precomputation step that occurs when the template is loaded. For each frame of the template and for each control particle, we estimate the control particle density gradient. For each control particle we then generate a repulsion particle at a distance h along the negative density gradient. Repulsion particles too close to a control particle are discarded. In practice, the distance threshold used is set to $a \cdot h$, with $a = 0.9$. Note that each repulsion particle k is oriented, defining its orientation $\mathbf{n}_k$ as the normalized control particle density gradient used to generate it.

Each of these repulsion particles generates a force in its neighborhood, pushing liquid particles towards the template surface, in the direction given by its orientation. Summing

up the repulsion forces exerted by repulsion particles on a liquid particle located at position $\mathbf{x}$ close to the template surface yields

$$f_r(\mathbf{x}) = w_r \sum_k \mathbf{n}_k W(d(\mathbf{x}, \mathbf{x}_k), h), \tag{3.3.4}$$

with w_r a constant scaling the strength of the repulsion force, $\mathbf{x}_k$ the position of k-th repulsion particle, $\mathbf{n}_k$ its orientation, and W the same kernel used for both attraction and velocity forces. Note that we could save a small portion of precomputation time by generating a repulsion particle only when the l^2-norm of the control particle density gradient is larger than a given threshold. In practice, this threshold is hard to find, leading to either too many particles created or not enough. We observed a smoother and denser repulsion particle layer when using the method described above.

3.3.3. Temporary Particles

The control forces defined earlier allow us to efficiently reproduce many templates when carefully placed in the scene by an animator. However in some cases, the kinematics of the control particles prevent the liquid to spread into regions covered by liquid in the template. In other cases, the amount of nearby liquid may not be sufficient to rapidly reproduce the whole template.

In order to be able to efficiently and robustly reproduce liquid templates in most cases, without considering whether it is even physically possible, we introduce *temporary particles*. A temporary particle is a liquid particle created by duplicating an existing liquid particle. Such a particle is removed when no longer useful, thus ultimately returning to conservation of mass. During its lifetime, the dynamics of a temporary particle is ruled by the Navier-Stokes equations and affected by our control forces.

At every frame and for every liquid particle in a controlled area, we compute the liquid particle density gradient $\nabla\rho$. For a liquid particle i located at position $\mathbf{x}_i$ and moving at velocity $\mathbf{v}_i$, we trace a position $\mathbf{x}_{tp} = \mathbf{x}_i - d_{tp}\frac{\nabla\rho}{\|\nabla\rho\|}$, with d_{tp} the distance between the liquid particle and the generated candidate, given by $d_{tp} = l \cdot h$. To generate our results, we used $l = 0.5$. We then estimate $\rho(\mathbf{x}_{tp})$ and $\rho_{cp}(\mathbf{x}_{tp})$, respectively the liquid and control particle densities. If the signed difference $\rho_{cp}(\mathbf{x}_{tp}) - \rho(\mathbf{x}_{tp})$ is larger than a given threshold, a temporary particle candidate is generated. During a cleaning pass, candidates are discarded if they are either too close to a liquid particle or to another temporary candidate. Each valid candidate is used to generate a temporary particle with the same physical properties than the particle it came from. Note that temporary particles can also emit themselves temporary

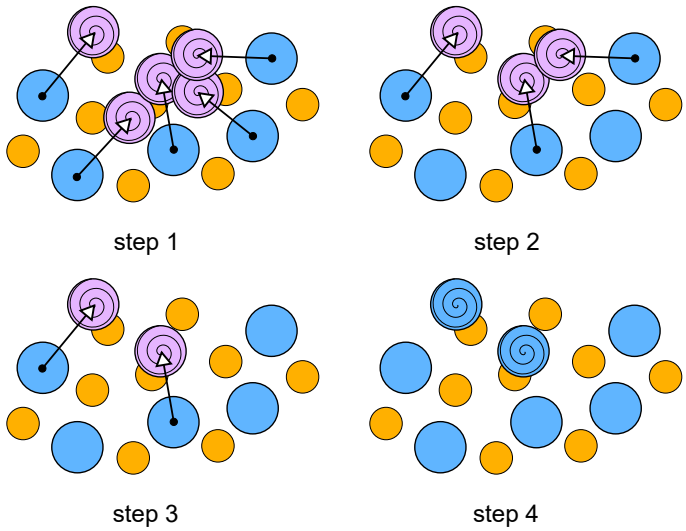


Fig. 3.3.4. Seeding temporary particles within a controlled area. Step 1: Each liquid particle (blue) generates a temporary particle candidate (purple, with a spiral pattern) if the density discrepancy between template (orange) and simulation is larger than a specified threshold. Step 2: Candidates too close to a liquid particle are discarded. Step 3: Candidates too close to another candidate are discarded. Step 4: The remaining candidates are transformed into temporary particles (blue, with a spiral pattern).

particles if needed. In practice, we disable the generation of temporary particle candidates too close to a solid. The seeding of temporary particles is illustrated in Figure 3.3.4.

Because we globally and ultimately want to respect the law of conservation of mass, we need to remove temporary particles when they become unnecessary. This happens when a temporary particle leaves every controlled area. However, this cannot be too sudden as it may lead to visual artifacts (e.g., a splash consisting mainly of temporary particles could disappear in mid-air as soon as it leaves a controlled area). In practice, we remove a temporary particle as soon as the density of its surrounding original (non-temporary) liquid particles is higher than a given threshold. The higher this threshold, the longer temporary particles will continue to interact with fluid particles outside every controlled area. Similarly to the creative process, we also enforce a minimal distance between candidates to deletion.

In practice, temporary particles extend the spectrum of possibilities from an artistic standpoint, and proved to be a simple, flexible, and efficient way to simulate physically inspired animations. Seeding and removing temporary particles respectively follow Algorithms 4 and 5.

Algorithm 4: Seeding temporary particles

```
Candidates = ∅;
for i ∈ Particles do
    emitted = FALSE;
    for s ∈ Instances do
        if x_i inside s then
            ∇ρ ← computeDensityGrad(x_i);
            x_tp ← x_i − d_tp ∇ρ/‖∇ρ‖;
            x_np ← getNearestParticlePos(x_tp);
            if ‖x_tp − x_np‖ < d_tp − ϵ then
                break;
            ρ ← computeDensity(x_tp);
            ρ_cp ← computeCpDensity(x_tp);
            if ρ_cp − ρ > η then
                Candidates = Candidates ∪ {(x_tp, v_i)};
                emitted = TRUE;
                break;
        if emitted then
            break;
for c ∈ Candidates do
    x_nc ← getNearestCandidatePos(c);
    if ‖x_np − x_c‖ < d_tp then
        Candidates = Candidates \ c;
for c ∈ Candidates do
    p ← generateParticle(x_c, v_c);
    Particles = Particles ∪ p;
    TemporaryParticles = TemporaryParticles ∪ p;
```

Algorithm 5: Removing temporary particles

```
Candidates = ∅;
for tp ∈ TemporaryParticles do
    if outsideEveryInstances(tp) then
        ρ ← computeNonTemporaryDensity(x_tp);
        x_nc ← getNearestCandidatePos(x_tp);
        if ρ > τ ∧ ‖x_nc − x_tp‖ > d_tp then
            Candidates = Candidates ∪ tp;
for c ∈ Candidates do
    removeParticle(c);
```

3.3.4. Animation-Editing Metaphors

Affine Transformation. In order to propose a comprehensive tool, we allow for time-varying affine transformations of our controlled areas. As mentioned earlier, standard translations, rotations, both uniform and nonuniform scalings, and reflections can thus be used and combined to modify template instances. In our system, each template instance contains a reference to the original template, as well as the affine transformation applied to it. Every time we want to evaluate a physical quantity of a transformed template instance at a given position and time, we apply the inverse transformation of the instance to this position in order to estimate this quantity into the untransformed (canonical) space. Control particle density is thus evaluated in the untransformed space. Similarly, attraction and repulsion forces are evaluated in the untransformed space, before applying the instance transformation to get the final forces. Velocity forces are computed the same way after transforming each liquid particle velocity using the inverse of each instance transformation. Estimation of density and forces is illustrated in Figure 3.3.5. In the case of an instance modified using a time-varying affine transformation, each control particle velocity is also virtually modified by summing the velocity of the instance at its world-space position.

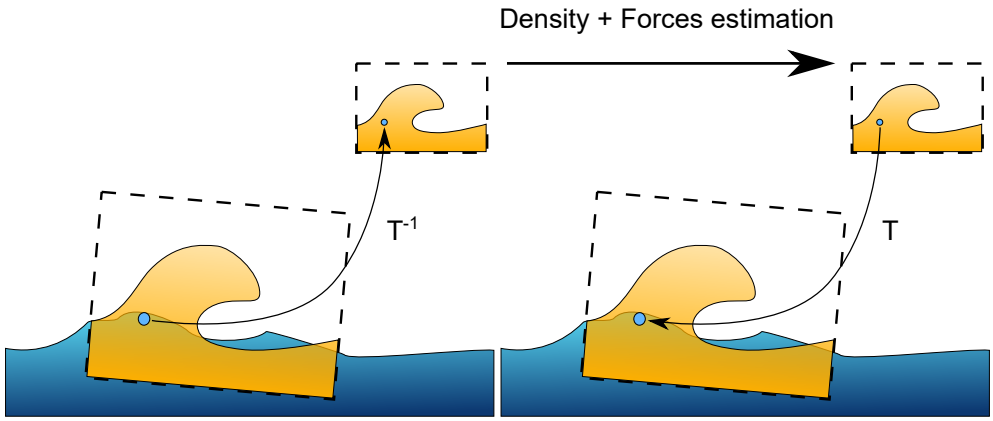


Fig. 3.3.5. Estimation of density and forces in a transformed template instance. The template instance is transformed by $\mathbf{T}$ in the scene. A position in the scene space is transformed by the inverse transformation $\mathbf{T}^{-1}$ in template canonical space to compute density of control particles as well as control forces.

Control Falloff. To reduce artifacts due to control discontinuities at the borders of a controlled area, we extend a falloff area around it. In this falloff area, temporary particles cannot be created nor removed. We also decrease control forces by replacing w_κ by

$$w_\kappa(t) = w_\kappa t^\gamma, \tag{3.3.5}$$

with $t \in [0,1]$ a linear coefficient defining the position along the cross section of the falloff area, $\kappa \in \{a,v,r\}$ the force type (respectively, attraction, velocity, and repulsion), and γ the decay parameter affecting how control is attenuated as soon as liquid particles leave the interior region of control areas.

Temporal Control. In our system, each template instance starts and ends affecting the simulated liquid at moments chosen by the user. In order to allow users to modify these temporal boundaries, we provide an animation-editing-like user interface. In our interface, each template instance is represented as a rectangle, similar to a track in an animation-editing software. Each instance can be temporally edited by translating or cropping the activation period of a template instance. In addition, a user can modify the play speed of a template instance, running the animation at half the speed of the original template for example. In such a case, we virtually modify each control particle velocity by multiplying it by the play speed factor. Finally and if a template instance is played in reverse order (i.e., from the end to the beginning), velocities of control particles are simply multiplied by -1 when evaluated. We believe that this familiar temporal animation-editing metaphor leads to simplicity and ease of use in global animation management.

Simulation and Template Frequencies. A template is captured at its animation speed with its own density of particles. The data is stored at every frame. When applied in a scene, a template can be scaled in space and time. Working in the original canonical space of a template allows for smoother transitions and correct physical quantities estimation. However, a user must be aware that in particular, temporal frequencies of a template cannot be scaled up or down too strongly (for example by a factor of 100 or more, or 0.01 or less) when applying it in a scene. During simulation, a frame of simulation will be computed as a certain number of time steps. A template generates its forces only once per frame, and its forces are applied at every simulation time step.

3.4. Implementation

Our methods for recording and compositing liquid simulations have been implemented in a system based on the open-source library SPlisHSPlasH [**10**]. Although we tested our liquid simulations on a number of SPH methods, we used DFSPH [**11**] to generate the results illustrated in this chapter and in the accompanying video. This choice is motivated by its stability as well as the small number of solver iterations required during simulation. We also

used the two-way coupling method of Akinci et al. [**1**]. Our graphical user interface, shown in Figure 3.4.1, is built on ImGui [**26**].

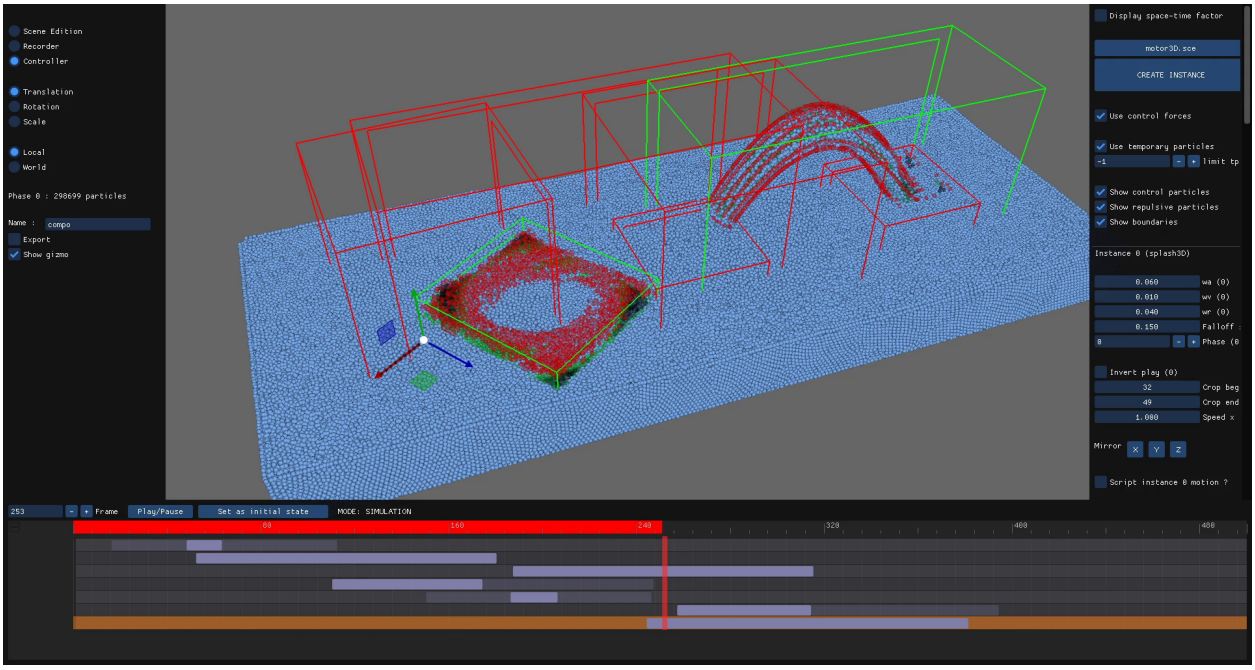


Fig. 3.4.1. GUI of our system. At the center, a 3D viewport allows the user to view the current state of the scene, including the controlled liquid, the rigid bodies, and the template instances created in the scene. Each template instance (green if active, red if inactive) can be transformed using a gizmo. The right panel of our interface can be used to create new template instances, and to modify parameters associated with each existing instance. The timeline at the bottom of the user interface is used to set when each template instance will start and stop affecting the liquid.

Once recorded, each template is stored as a binary file. In our compositing tool, templates are always loaded once in memory, even when instantiated several times in the scene. During simulation and in order to speed up particle search, we use several uniform acceleration structures usually called buckets in the literature. Two buckets covering the whole simulation domain are used to store liquid particles and temporary particle candidates. In addition to those buckets, two local buckets covering each template are used to speed up control and repulsion particle search. When loading a template, control and repulsion particles are sorted in those buckets for the whole template duration, allowing efficient particle neighbor searching at runtime.

During the scene (i.e., liquid and rigid bodies) simulation, the state of each element of the scene is stored at each frame in memory, in order to allow fast replay of an art-directed

animation. This is particularly useful to visualize and appreciate the complex motions of liquids, after slower editing in large 3D domains. Our system is a multi-threaded CPU implementation using OpenMP.

To design our 3D examples, we initialized the weights of our forces using the following set of default values: $w_a = 0.06$, $w_v = 0.01$, $w_r = 0.04$. We then slightly refined those parameters during a short iterative process. In 2D, the initial set of parameter values used is $w_a = 0.2$, $w_v = 0.03$, $w_r = 0.1$. For all the examples, we used $\eta = \tau = 0.35\rho_0$ as density thresholds, with ρ_0 the rest density.

3.5. Results

3.5.1. 2D Proof of Concept

We used our system to generate many controlled liquid animations. As a proof of concept and in order to illustrate a number of aspects of our system, we first tested it on 2D examples. These 2D examples are simulated mostly at interactive framerates, ranging from 5.3 to 30.8 frames per second (see column *Simulation* in Table 1). Note that we use DFSPH [**11**] to perform the simulation. We show images of particles and bounding boxes in 2D in order to better display various features of liquid and control particles, templates, etc. In the following images, liquid particles are shown in light blue, control particles in a color gradient ranging from green (for α close to 0 in Equation (3.3.2)) to black (for α close to 1), and repulsion particles in red. On the images depicting a fluid being recorded, the red bounding box corresponds to the recording region. On the images representing a liquid simulation controlled using a template, the inner green bounding box represents the boundary between the falloff and the fully controlled area. The outer green bounding box represents the boundary between the uncontrolled and the falloff area.

2D Rectangular Drop. In a first simple example depicted in Figure 3.5.1, we recreate the impact of a rectangular drop of water falling in a pool. We wanted to show the impact of a drop of water, although without inserting new particles that would need to appear in mid-air. Our method correctly modifies the liquid behavior of the simulation to locally recreate the drop impact and propagating ripples, without visual artifacts due to the drop during its fall.

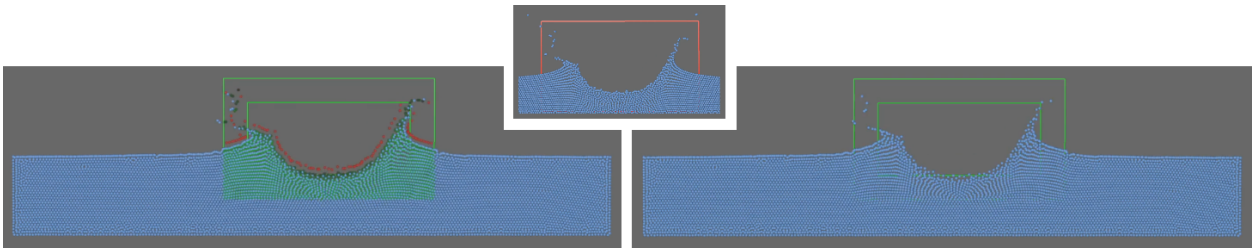


Fig. 3.5.1. A 2D liquid simulation controlled using a template describing the impact of a rectangular drop of water on a liquid surface. Top: Recording of the template. Left: The template is instantiated on a calm surface of water to recreate the impact of the drop. Right: Same animation with template visualisation disabled.

We also instantiated this template before changing its scale. On the example depicted in Figure 3.5.2, we controlled a calm surface of water using the same template, but rescaled respectively by a factor of 2.0 and 0.5, along the Y and X axis. Estimating the physical quantities in the canonical space allows us to robustly reproduce the template transformed by a user.

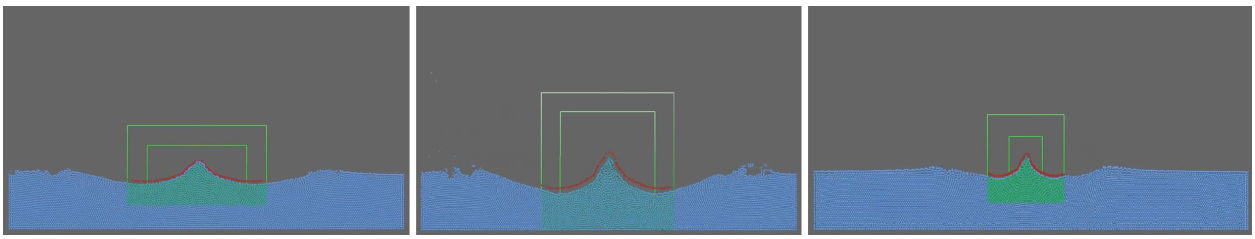


Fig. 3.5.2. Other 2D liquid simulations controlled using the same template than in Figure 3.5.1, but rescaled along one axis. From left to right: Original template, template scaled by a factor of 2.0 along the Y axis, template scaled by a factor of 0.5 along the X axis.

2D Wave Machine. We recorded a wave template from a wave-machine-based simulation, created by using a box containing a liquid and oscillating with a given frequency and amplitude. Once instantiated in the scene, the template is correctly reproduced as shown in

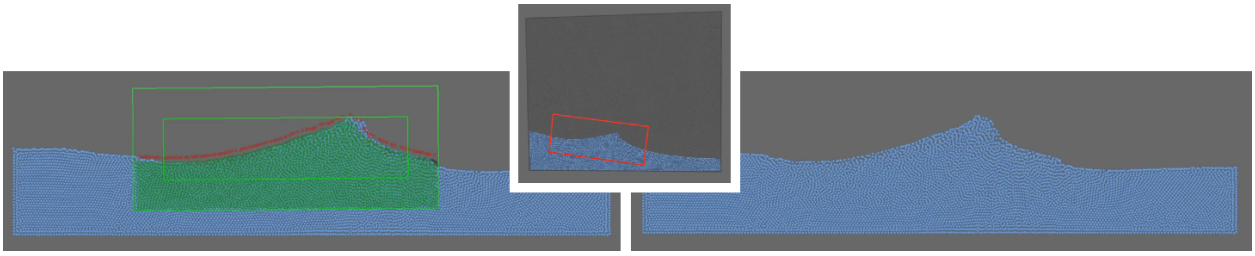


Fig. 3.5.3. A 2D liquid controlled using a wave template. Top: A wave template recorded using a wave machine. Left: A liquid simulation controlled using the resulting template. Right: Same animation with template visualisation disabled.

Figure 3.5.3, giving birth to a wave that propagates out of the controlled region in a realistic way. Thanks to the falloff area, boundaries are smooth between controlled and uncontrolled liquids, leading to a convincing blended result.

2D Rotating Blade. Our method is compatible with templates containing solid objects. Thanks to its layer of repulsion particles, our method provides stronger control in boundary-liquid template regions, correctly reproducing these regions. We demonstrate this aspect in a simple 2D example illustrated in Figure 3.5.4. We instantiate a template storing the behavior of a liquid mixed by a rotating blade. Not using repulsion particles shows a smoother relative displacement of liquid particles, while using them follows more closely the blade boundaries.

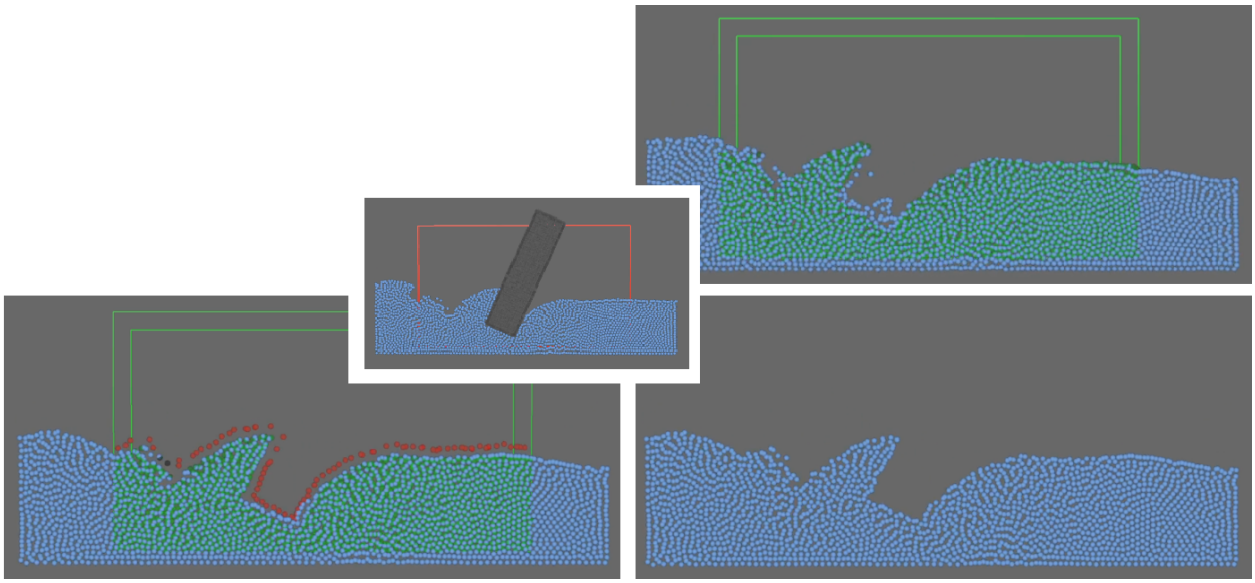


Fig. 3.5.4. Template of a 2D liquid mixed by a rotating blade, used to control a liquid. Center: Recording a portion of a 2D liquid mixed by a rotating blade. Bottom left: The template is used to control a calm surface of water. Thanks to the layer of repulsive particles, the template interface is correctly reproduced. Bottom right: Same animation with template visualisation disabled. Top right: As a comparison, we performed the same control without using repulsion particles.

2D Water Jet. In another example shown in Figure 3.5.5, we recorded a portion of a thin jet of water created using two rigid bodies forming a funnel. The resulting template is then instantiated twice to control a liquid, directing the jets respectively from the bottom left to the top right, and from the bottom right to the top left. Our system controls the liquid, creating two jets of liquid that gush from the surface and collide in the air.

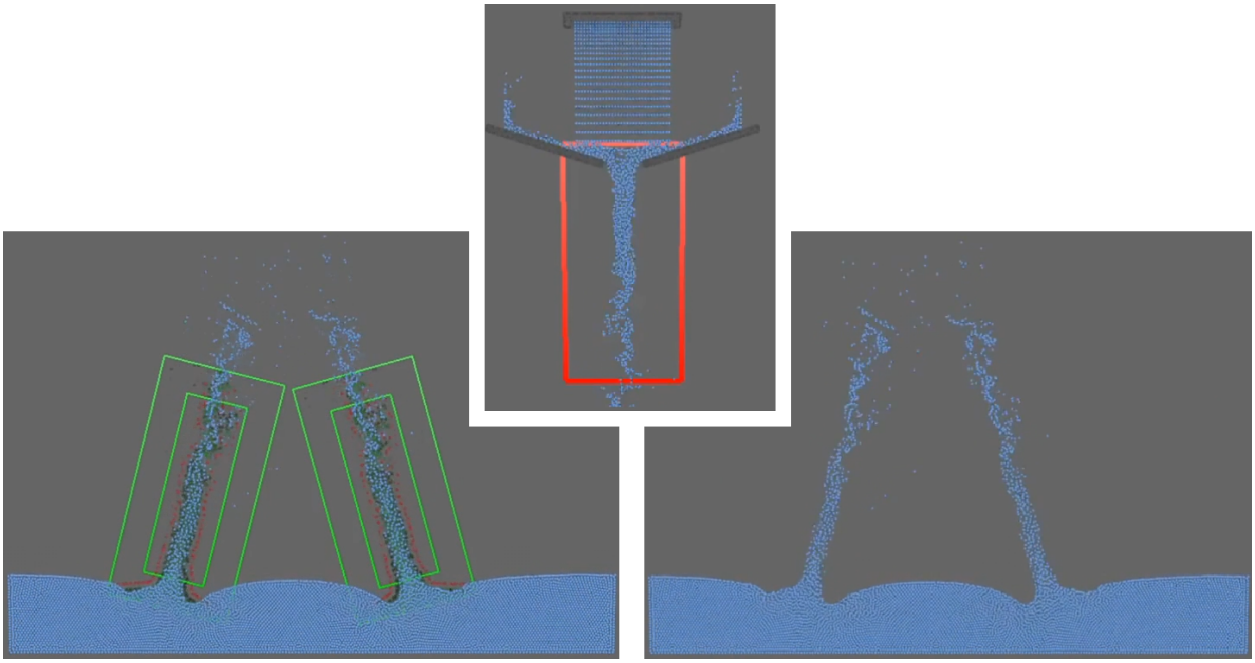


Fig. 3.5.5. Simulation controlled using instances of a jet template. Top: Recording a trickle of water flowing through a funnel formed by two rigid bodies. Left: The resulting template is instantiated twice to make two jets gush from a calm surface of water. Right: Same animation with template visualisation disabled.

2D Zero Gravity.

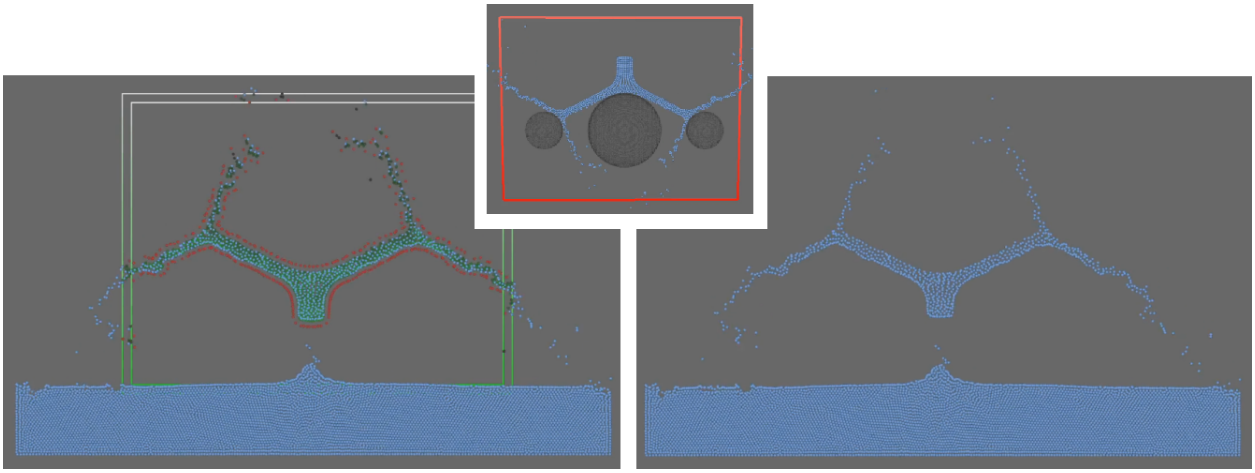


Fig. 3.5.6. 2D liquid simulation controlled using a template of a liquid not affected by gravity. Top: A jet of water is recorded, not affected by gravity and splitting while hitting rigid bodies. Left: The recorded template is used to control a liquid, this time affected by gravity. Right: Same animation with template visualisation disabled.

We recorded a 2D template in which a thin water jet, unaffected by gravity, evolves through the simulation domain while hitting disks as obstacles. Although this scenario shown in Figure 3.5.6 is unrealistic, we can instantiate it in a scene, flip it upside-down, to faithfully reproduce the behavior of the template, even under gravity. Note how the repulsion particles (in red) help to channel and constrain the liquid particles where the jet has a sharper surface.

3.5.2. Going to 3D

Our 3D examples are simulated at lower framerates, due to much larger sets of liquid particles. They go from 0.8 to 48.5 seconds per frame (see column *Simulation* in Table 1). This is where an extension to GPU support would come handy (see Section 3.6).

3D Rotating Blade. Several 3D simulations have been generated using our method. Figure 3.5.7 shows a scene where a calm surface of water is perturbed with a template describing the dynamics of a liquid mixed by a blade rotating horizontally. Similarly to our 2D experiment shown in Figure 3.5.4, our system efficiently and correctly reproduces the template in both space (region covered by the template instance) and time (over the duration of the template). Liquid particles spin away from the center, creating a hole and partly-circular waves.

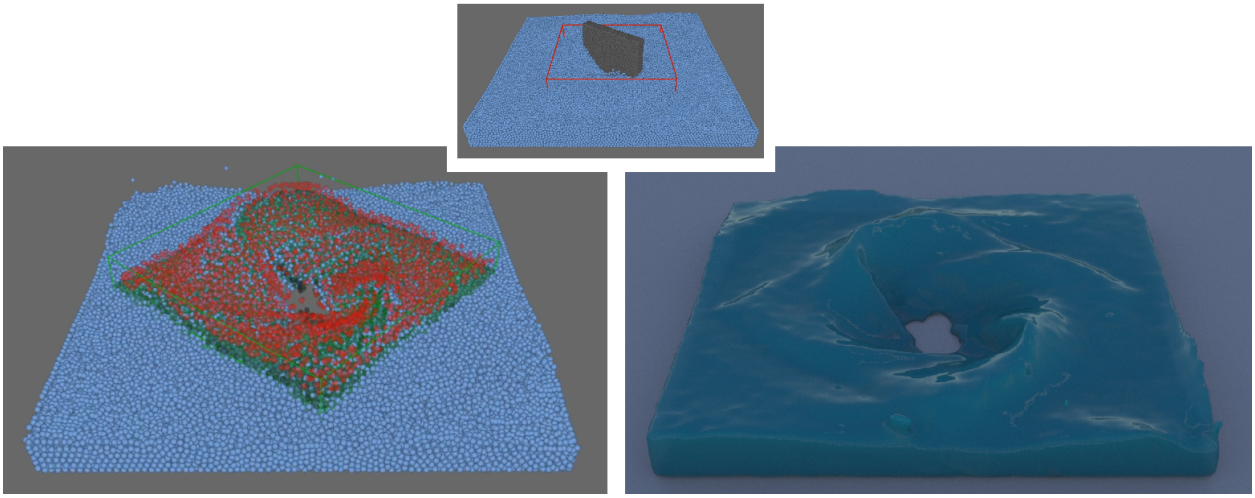


Fig. 3.5.7. 3D liquid controlled using a template describing a liquid mixed using a 3D rotating blade. Top: Recording a portion of a liquid mixed using a rotating blade. Left: Liquid simulation controlled using the resulting template. Right: Offline rendering of the resulting liquid surface.

3D Viscous Liquid. Our method is compatible with templates simulated under physical properties different than the controlled liquid. In Figure 3.5.8, we control a water-like liquid to follow a buckling behavior recorded from a viscous honey-like liquid. The template is scaled by a factor of two in all three dimensions and flipped upside-down in order to make a buckling pattern magically rise from the water surface. Although viscous liquids and water exhibit very different behaviors, we can still control a water simulation, creating the buckling pattern in mid-air.

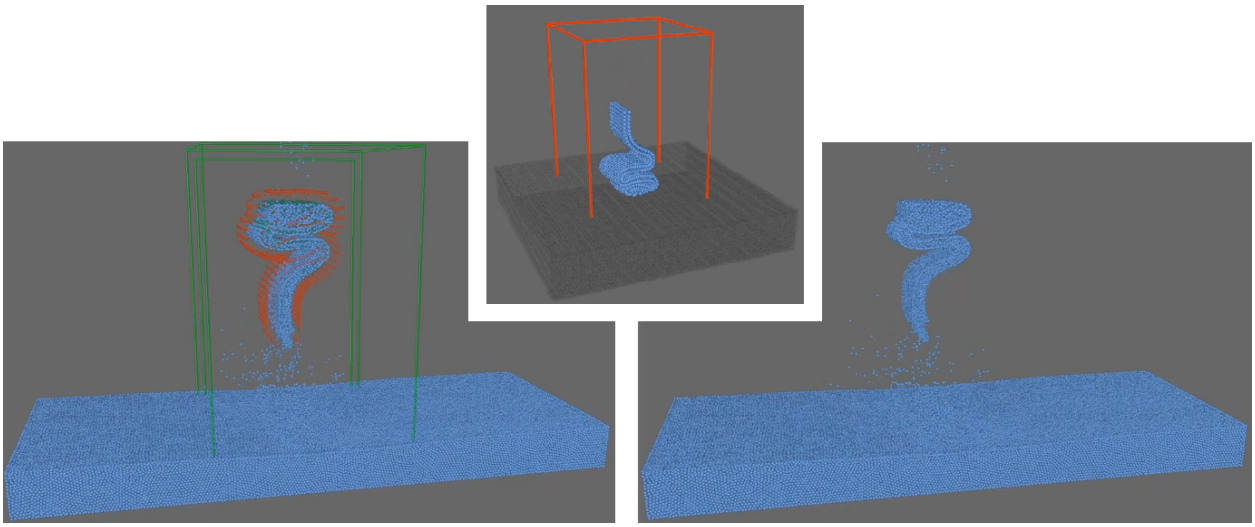


Fig. 3.5.8. A viscous 3D liquid template is used to control a liquid simulation. Top: Honey-like liquid recorded while forming a buckling pattern. Left: This template is flipped, rescaled, and used to control a water-like liquid. Right: Same animation with template visualisation disabled.

3D Letters. When loaded in memory, templates are played in their usual time-forward direction by default. However, users can play a template in reverse order. This feature is demonstrated in a 3D example where we recreate the shapes of original meshes with a controlled liquid. During a recording phase, water-filled letter shapes fall and splash at the

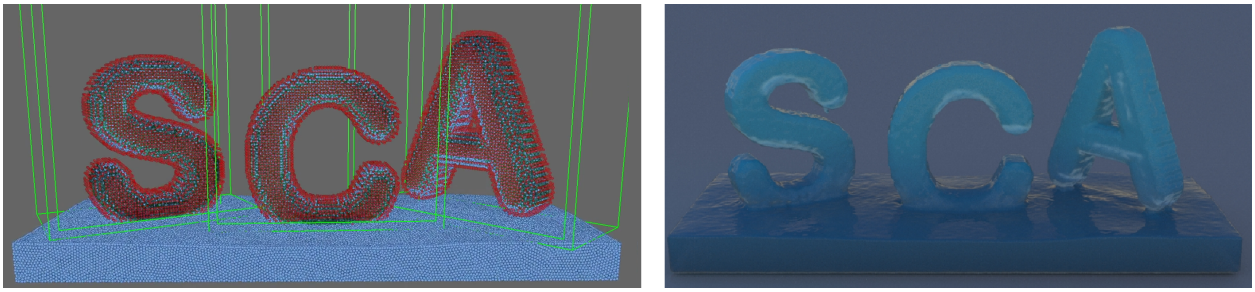


Fig. 3.5.9. Liquid controlled to form the "S", "C", and "A" letters in 3D. Left: Three templates, one for each letter, are instantiated in the scene and played in reverse order when controlling the liquid. Right: Offline rendering of the resulting liquid surface.

bottom of the simulation domain. In those simulations (one for each letter), we slightly reduced gravity in order to lead to smaller splashes. After recording the simulations, we instantiated the templates in a scene, playing them from the end (liquid at the bottom of the simulation domain) to the beginning (liquid forming letters). Our method efficiently reproduces the animation, drawing water from the overlapping simulation domain to recreate the letters. A frame from this scene is illustrated in Figure 3.5.9.

3D Multiphase. Our system is compatible with the control of multiphase liquids. When a template is instantiated in the scene, a user can select which liquid phase the template instance controls. Figure 3.5.10 shows an art-directed simulation generated using our method. Two instances are generated in the scene (one being scaled along the Y axis), each of them controlling a different phase of a two-phase liquid. The two liquids are pulled out of their respective spaces, jump in mid-air, and dive back in the liquids, partially mixing at the bottom of the reservoir with an expected behavior.

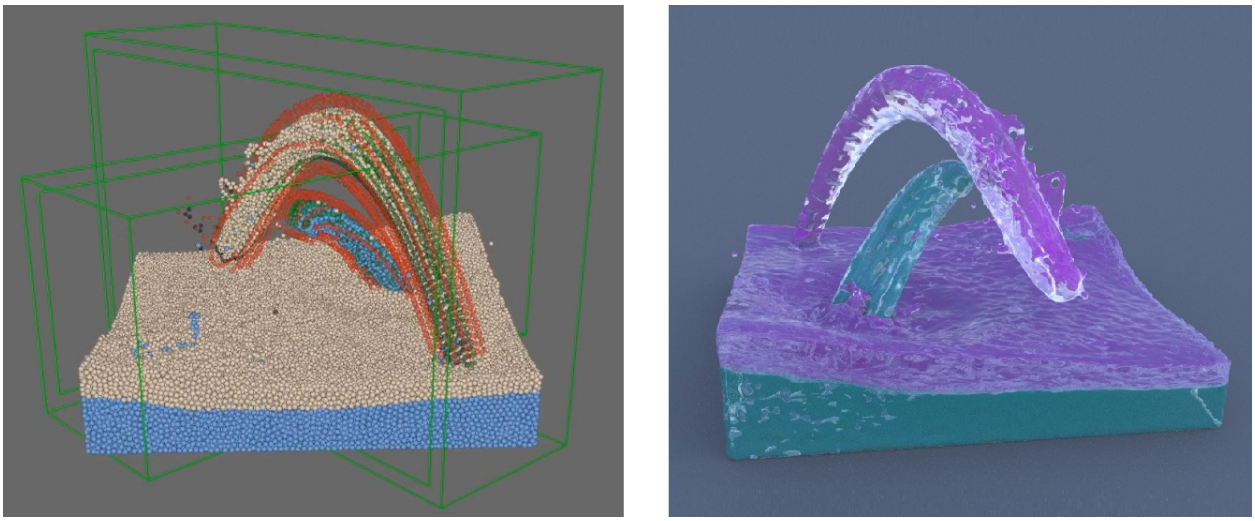


Fig. 3.5.10. 3D multiphase liquid control. Left: A template describing the behavior of a jet of water is instantiated twice in the simulation domain, each instance controlling one phase of the liquid. Right: Offline rendering of the resulting liquid surface.

3D Hand. In our system, a controlled liquid can interact with rigid bodies. In Figure 3.5.11, a hand made of water raises a floating boat in mid-air. With a strategy similar to the one from Figure 3.5.9, a water-filled hand shape has been recorded, falling and splashing in an empty scene. This template is played in reverse order while controlling the liquid, in order to make a hand emerge from underneath. In this simulation with around 2.49M particles, the water from the hand interacts with the boat using classical two-way coupling.

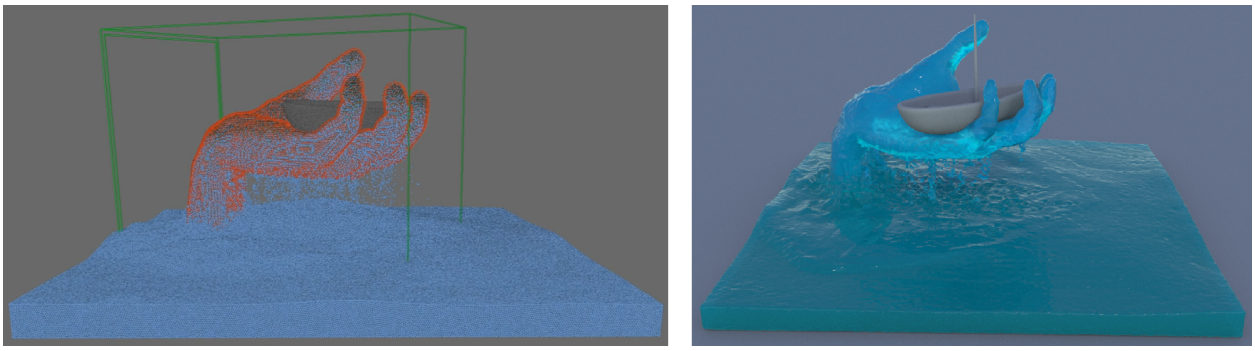


Fig. 3.5.11. Poseidon's hand raising a boat above the surface of the sea. Left: A template describing the behavior of a hand, made of water, splashing in the scene, is played in reverse order to make it emerge from the liquid. Right: Offline rendering of the resulting liquid surface.

3D Composition. In this last example, we combined several templates, each of them starting and ending to control the simulated liquid at user-defined times. In the final animation, a jet of water jumps in mid-air twice, before diving back in the liquid in a whirlpool. Our graphical user interface inspired by video editing tools made the design of the sequence very simple. Two images from this animation are shown in Figure 3.5.12.

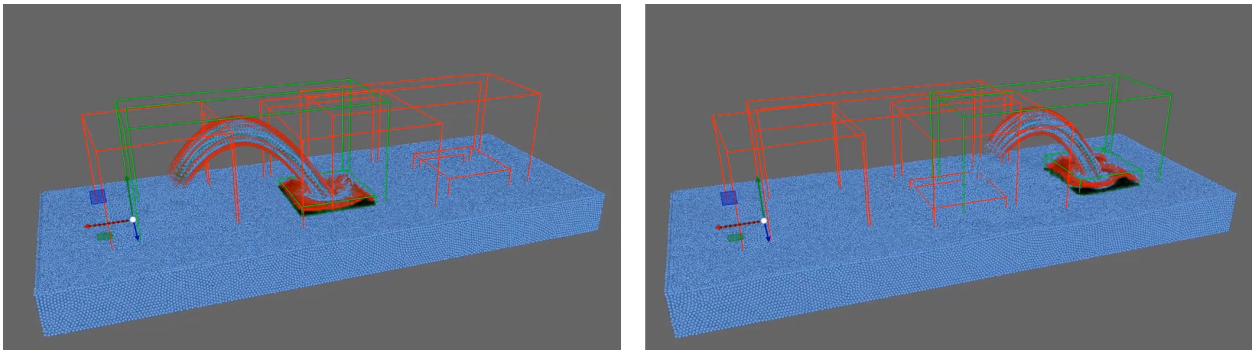


Fig. 3.5.12. Two images of an animation showing a jet of water jumping and diving back in a whirlpool. This animation has been created by combining a number of templates using our system.

3.5.3. Statistics

Table 1 gives some statistics about the simulations showcased in this section. All the simulations were run on a desktop PC with an i9-9900K 8-core CPU at 3.60 GHz with 128 GB of memory.

	Particles (avg per frame)				Timings (ms per frame)		Memory	Storage
Scene	Liquid	Control	Repuls	Tempo	Sim	Control	GB	MB
Fig. 3.5.1	6.7k	1.3k	88	36	190.2	5.6 (2.9%)	0.33	3.87
Fig. 3.5.3	6.7k	2.3k	130	14	128.7	13.7 (10.6%)	0.34	7.44
Fig. 3.5.4	2.2k	1.3k	79	0	32.4	6.7 (20.7%)	0.24	12.0
Fig. 3.5.5	6.7k	1.3k	217	138	96.7	3.9 (4.0%)	0.35	4.19
Fig. 3.5.6	6.7k	0.6k	209	0	116.5	3.4 (2.9%)	0.46	3.02
Fig. 3.5.8	298.7k	1.3k	713	76	6566.5	53.2 (0.8%)	5.81	5.11
Fig. 3.5.7	56.2k	20.0k	2.3k	0	807.2	135.4 (16.8%)	2.4	82.7
Fig. 3.5.9	298.7k	39.8k	8.2k	329	5207.5	474.8 (9.1%)	7.4	274.1
Fig. 3.5.10	106.4k	2.6k	1.4k	74	2274.5	125.7 (5.5%)	3.2	5.12
Fig. 3.5.11	2.49M	363.3k	30.1k	994	44585.2	3921.6 (8.8%)	80.6	3488.0
Fig. 3.5.12	298.7k	12.2k	1.7k	43	4900.1	112.4 (2.3%)	7.8	90.0

Table 1. Various statistics about our controlled simulations.

The numbers of control, repulsion, and temporary particles are given as an average per frame, for all frames with at least one template activated. These particles are not included in the number of particles in the *Liquid* column, the latter being constant within a simulation. The number of control particles is the average of particles stored in its template. This number, multiplied by the number of frames for the duration of its template, gives the relative size of the file stored on disk, shown in column *Storage*. The ratios Control/Liquid particles vary from 9.0% to 59.1% in 2D, and 0.43% to 35.6% in 3D. The number of temporary particles per scene is relatively low, but proved important in special cases with high discrepancy between the coverages template-scene. The number of repulsion particles depends on the proportion of surfaces in the template.

Timings are given as an average per frame during an activated template. Control is applied once per frame, while simulation with the DFSPH [**11**] method requires an adaptive number of steps. Control in 2D adds from 2.8% to 20.7% to the simulation time, and in 3D from 0.8% to 16.8%.

The *Memory* column includes all particles, templates, acceleration structures, recorded simulation to replay in real time, and the system itself, with its libraries. The system without any liquid nor template occupies 71.2 MB.

3.6. Conclusions and Future Work

Liquid animations are notoriously difficult to design. Indeed, a fully physical simulation offers too little control if an artist can only set initial configurations and physical properties, while a fully artist-based animation may not look as realistic. We described our intuitive tool that allows artists to compose SPH-based liquid simulations using precomputed liquid simulation templates. Our system generates forces from control particles issued from animation templates in order to drive a global liquid simulation. When desired, temporary particles are automatically and seamlessly added and removed when insufficient liquid can fill empty regions. Our system is easy to use, robust, efficient, and generates predictable animations. To demonstrate its potential, we created and analyzed several animations that would have been very complex to generate without our method. In practice, our system could be used as part of a first interactive editing stage, followed by a second liquid upresolution stage (for example using the methods from Nielsen et al. [**81**] or Mercier et al. [**72**]) as described in Figure 1.1.3.

While we are generally happy with our prototype system, there are several improvements to make it even more efficient and robust. We used basic acceleration structures and multithreading on CPU to reach interactive results on simple simulations. Our system adds little overhead on top of our SPH simulation. We have not yet looked at going from CPU-only to GPU, but it should deliver reasonable gains since our control strategy can be easily and efficiently parallelized.

Our system is not designed to robustly handle cases where multiple overlapping template instances would be used to cover the same region of space. In its current state, our system computes the resulting control force by summing the forces generated by each instance. This strategy does not provide a satisfying result in some cases (e.g., a template instantiated twice at the same location with the exact same deformation). To tackle this issue, we could interpolate between the forces generated by each template instance. Similarly, this is not a good solution in many cases (e.g., a template instantiated twice with the exact same deformation, but one mirrored). The method should be improved to robustly handle this type of scenario. Additionally, more complex ways to combine templates could also be found, allowing a user to subtract templates or attach them to some fluid features.

While breaking physical correctness, the notion of temporary particles has provided flexibility to follow more closely some template simulations. However, because we rely on local considerations to create and remove temporary particles, we are not globally optimal. Using

global considerations or optimizing over a wider time window would lead to more optimal solutions, including fewer and visually less noticeable temporary particles, but at an increased computational cost.

Data compression could be relevant to efficiently store and load templates involving hundreds of thousands particles.

The placement of template instances is sometimes a little challenging in order for them to be seamlessly integrated in a liquid. Automatic position and refinement of control parameters over time could be investigated to improve this aspect.

Our system is not designed to handle transformations involving large scaling factors, and side effects can appear. Our method should be improved to robustly handle these extreme deformations. For now, template boundaries are also limited to boxes. We could easily extend our system to make it compatible to a wider range of inner and outer template boundary shapes, as well as morphing boundaries according to the surrounding particles and obstacles.

Other directions for future investigations include editing other types of fluids (e.g., smoke or fire), and applying our method to multiresolution flows. Extensions to other types of materials (e.g., granular materials or snow) could also be investigated. Since we are highly interested in art-directed control, performing a user-study with artists specialized in fluid animation would be relevant.

Chapter 4

A Density-Accurate Tracking Solution for Smoke Upresolution

4.1. Introduction

Generating art-directed high-resolution fluid simulations is a complex and tedious task. Using a set of specialized tools, an animator can control a high-resolution fluid simulation. However, this process leads to long iterations of the creative process, since a full high-resolution resimulation is required every time the animator wants to visualize an intermediate result. As mentioned in the introduction of this thesis, a naive solution to this problem is to control a low-resolution model, and once targeted results are obtained, to generate the final simulation using a finer grid. This strategy does not lead to satisfying results, in the

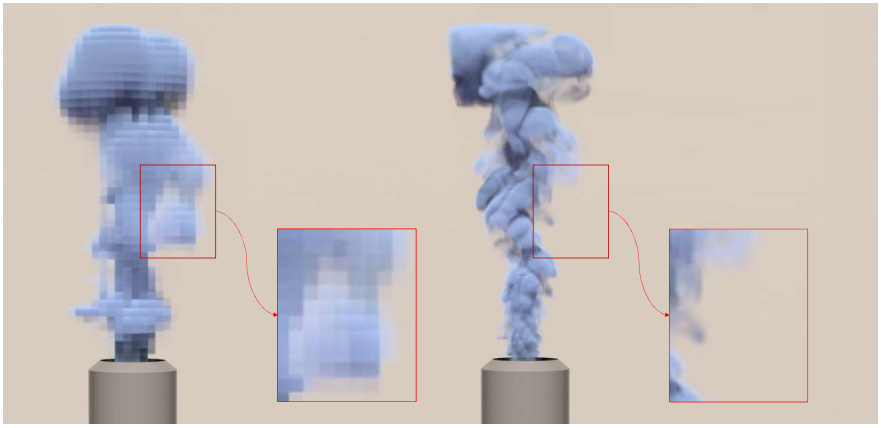


Fig. 4.1.1. Increasing the resolution of the simulation grid changes the coarse aspect of the smoke. Left: Smoke simulated using a $32 \times 48 \times 32$ grid. Right: Same frame of a smoke simulation generated using the same simulation parameters and initial conditions, but using a finer grid of resolution $128 \times 192 \times 128$.

sense that the final high-resolution simulation usually does not reproduce the coarse look of the low-resolution simulation, as shown in Figure 4.1.1. The mismatch can be explained by

several factors including the change of artificial viscosity, or the variation of results obtained after performing a pressure projection. In practice, changing the resolution of a simulation leads to unexpected behaviors.

The mismatch obtained when changing the resolution of a simulated model is the *raison d'être* of upresolution techniques. Taking a low-resolution simulation as an input, these techniques generate a high-resolution simulation following the coarse aspect of the low-resolution simulation but enhanced with fine details specific to high-resolution simulations.

Inspired by several methods including optimization-based control methods [**125, 71, 88**], we propose a new way of tracking a high-resolution smoke simulation using a low-resolution simulation, built on optimization-based density matching. Where previous methods failed to reproduce a density distribution similar to a guide, our method specifically constrains high-resolution smoke to match the coarse density distribution of the guide, while giving birth to fine simulated details. Specifically, our method enables artists to use a low-resolution dense sequence of density distributions to guide an Eulerian smoke simulation at a higher resolution. In our method, the velocity field of the high-resolution simulation is optimally modified in order to increase the density matching with the low-resolution simulation, leading to more predictable and plausible results. For additional flexibility, an artist can reduce tracking fidelity, giving more freedom to the high-resolution simulation.

We have evaluated our method on several scenarios, different in terms of domain size, animation duration, tracking parameters, and obstacles. We have applied it to enrich data from real-world dynamic smoke with physically simulated details. Our method proved to faithfully and automatically reproduce guide simulations of smoke density distributions. We have also generated simulations from less realistic animations, for example by combining two simulations into a third one, thereafter used as a guide. As such, our method introduces a basis for tools for artists to generate plausible controlled smoke animations.

Note that the completion of this project resulted in a paper [**106**] published at the journal *The Visual Computer*, presented at the conference *Computer Graphics International* 2020. An accompanying video can be found on the webpage (`http://arnaud-schoentgen.com/publication/smoketracking/`) dedicated to this project.

The rest of this chapter is organized as follows. We first discuss related work in Section 4.2. We describe our method in Section 4.3 before presenting results in Section 4.4. Conclusions and future challenges are drawn in Section 4.5.

4.2. Related Work

As discussed in the previous chapters, researchers have proposed several strategies to generate a high-resolution smoke simulation from a low-resolution one. Procedural techniques such as from Kim et al. [**60**] use a procedural turbulence function to increase the complexity of a low-resolution velocity field with fine details. The generated procedural perturbations are divergence-free and located in a spectral band guaranteeing the preservation of existing structures. This approach is efficient and easy to implement. Unfortunately, the procedural nature of the generated details may lead to a lack of plausibility of the resulting animations, especially for large upresolution factors. Note that Gregson et al. [**39**] applied this method as a final *beautification pass* after a guiding stage. In Section 4.4, we provide a comparison between our method and the one of Kim et al. [**60**].

During the past few years, learning-based techniques have been proposed to synthesize a high-resolution version of a coarse simulation using neural networks trained on sets of simulation data [**23, 133, 128, 132, 5**]. This type of approach is very promising, and methods that become more physics-aware deliver better results. However, large amounts of data and long computation times are required for training. Moreover, the quality of the results depends highly on the data used for training. As a consequence, the system thus has to be re-trained in case of a modification of simulation parameters. Furthermore, divergence-free motions cannot be guaranteed.

Some techniques propose to guide a high-resolution simulation using a low-resolution input simulation. Nielsen et al. [**83**] and Nielsen and Christensen [**82**] solve a minimization problem prescribing that the low-frequency components of the simulated velocity field should be as close as possible to the guide velocity upsampled at high resolution. In practice, density variations between the simulation and the guide can be observed after a few advection steps, and they accumulate over time. As mentioned by the authors, this mismatch is mainly due to the variation of density diffusion and density dissipation depending on the resolution. We provide a comparison with this approach in Section 4.4. In order to guide a smoke simulation using tomographic scanning, Gregson et al. [**39**] use a proximal method to minimize an objective function containing both a divergence-free term and a tracking term. The latter term is, in turn, defined as the sum of three terms: an optical-flow photoconsistency, a smoothness, and a kinetic energy penalty. Taking a low-resolution smoke simulation as an input, Huang et al. [**46**] sample the simulation domain using match points, before using a correction force to reduce an error estimated at this position. This approach is efficient but the quality of the guiding depends on the strategy used to sample the simulation domain.

Sato et al. [**99**] propose to solve a minimization problem in a stream function space. Using a flow representation based on stream functions guarantees their incompressibility. Note that we discuss the use of such a parametrization to solve our problem in Section 4.3.6.

Several methods [**83, 82, 49**] constrain the low-frequency components of the velocity field of a smoke simulation to be similar to a target velocity field. As we mentioned, these methods usually fail to accurately reproduce the coarse aspect of the density distribution of the guide simulation. The volume of densities being what we do visualize when rendering smoke, we believe that faithfully reproducing it is crucial when tracking a smoke simulation. Bergou et al. [**12**] introduce TRACKS, a method for deformable surface upresolution where low- and high-resolution discretizations of a same surface are decomposed into patches with a pairing between low- and high-resolution components. The method constrains each high-resolution patch, in order for its center of mass to match the center of mass of its low-resolution counterpart, while being enhanced with physically simulated details such as foldings and wrinkles.

4.3. Our Method

4.3.1. Overview

Our goal is to use a lower-resolution simulation (called *guide* simulation) to guide a smoke simulation (called *tracked* simulation), respecting the same global movements and visual appearance while generating physically-plausible smaller-scale details. Unfortunately, a direct re-simulation performed at a higher resolution with the same set of parameters and initial conditions but without any control (called *free* simulation) can lead to unpredictable behaviors, as shown in several examples throughout Section 4.4. Our proposed method is designed to allow artists to ultimately edit more easily a coarse simulation in order to shorten the art-direction process, confident that a higher-resolution simulation adapted to the artistic vision could be generated.

In designing our controller used to perform smoke tracking, we are guided by three main considerations:

(1) Once advected through the controlled velocity field, we want the tracked simulation to be as close as possible to the guide simulation in terms of density distribution.

(2) The overall control applied to the tracked simulation should be minimized in order to penalize solutions leading to an over-controlled simulation.

(3) The velocity field modification applied to the tracked simulation must be divergence-free in order for the simulation to conserve mass over time.

These requirements give rise to several challenges, addressed by casting the problem as an optimization

$$\boldsymbol{u}^* = \underset{\boldsymbol{u}, \nabla \cdot \boldsymbol{u} = 0}{\arg\min}\, \phi(\boldsymbol{u}), \tag{4.3.1}$$

solving for an optimal divergence-free velocity modification $\boldsymbol{u}^*$ with an appropriately chosen objective ϕ. The specifics of the objective matter much, and we discuss them in the following sections. Using a continuous optimization solver, we require a spatially-continuous and differentiable advection scheme in order to compute optimal velocity perturbations (Section 4.3.5.) Furthermore, to achieve reliable density tracking without over-constraining the high-resolution simulation, we must distinguish between fine- and coarse-scale components of the density field in a computationally efficient way (Section 4.3.3).

Throughout the chapter, we refer to the density distribution $\boldsymbol{\rho}$ of simulation k at time t as $\boldsymbol{\rho}_k^t$, with $k \in \{G, T\}$. Here, G and T refer respectively to the guide and the tracked simulation. Note that $\boldsymbol{\rho}_k^t$ is a fairly large vector that contains all densities of the grid stacked on top of each other. Similarly, we refer to the velocity field $\boldsymbol{v}$ of simulation type k at time t as $\boldsymbol{v}_k^t$. We refer to the blur operator as $\mathbf{B}$, to the projection operator as $\mathbf{P}$, and to the advection of field A through vector field $\boldsymbol{v}$ as adv(A,$\boldsymbol{v}$).

4.3.2. Smoke Simulation

Our smoke tracking framework is based on Stable Fluids [**115**]. In a standard Eulerian smoke simulation, the density and velocity states are iteratively updated using a splitting scheme on the Navier-Stokes equations. We propose to augment the standard simulation loop by inserting an extra *tracking* step, after the projection step and before the advection step. For each simulation time step, our tracking step adds a velocity field modification $\boldsymbol{u}^*$

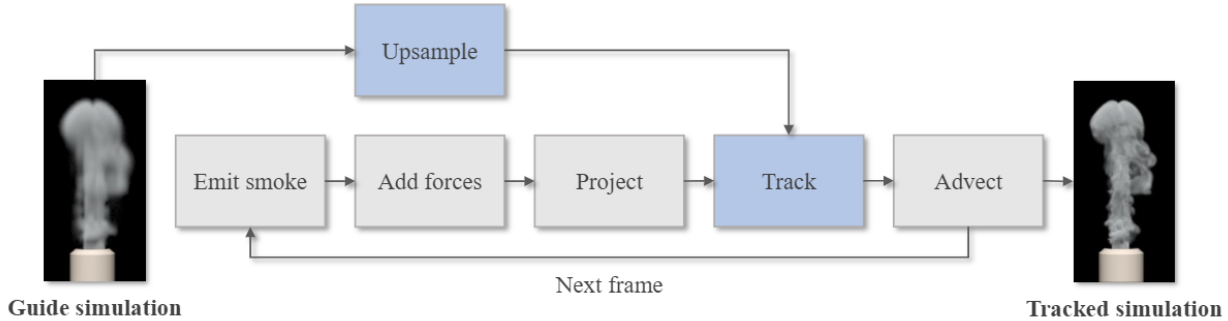


Fig. 4.3.1. Schematic overview of our tracking method. In blue are our added steps over a standard Eulerian smoke simulation.

to the projected velocity field of the tracked simulation, such that its density field becomes as close as possible to the corresponding low-resolution field after advection. The resulting velocity field must be divergence-free, and so, we constrain the solution of the optimization to be divergence-free. Figure 4.3.1 gives a schematic overview of the augmented standard smoke simulation loop for our method.

4.3.3. Density Tracking and Blur

For each simulation step t, the introduced tracking step solves for the velocity field modification $\boldsymbol{u}^*$, minimizing the difference between the advected tracked density $\boldsymbol{\rho}_T^{t+1} = \text{adv}(\boldsymbol{\rho}_T^t, \boldsymbol{v}_T^t + \boldsymbol{u}^*)$ and the next guide density $\boldsymbol{\rho}_G^{t+1}$. Unfortunately, simply minimizing den-

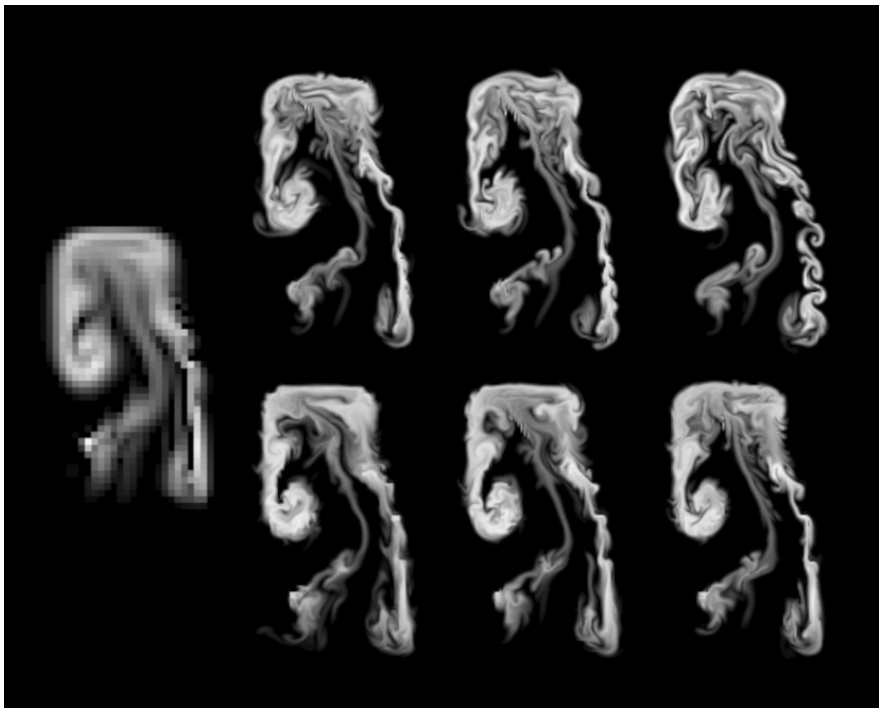


Fig. 4.3.2. Visual impact of the blur radius. Left: Guide. Right: Tracked with blur radius $r \in \{0, 2, 4, 6, 8, 12\}$, in the usual order. Note that r is given in terms of high-resolution grid cell size, with an upresolution factor of 4.

sity differences leads to several issues. Since the guide density field has a lower resolution than the tracked simulation, it obviously lacks information to fully determine the desired high-resolution density distribution. Using a frequency argument, we can estimate that the guide tells us what the low-frequency components of the final density distribution should be. To only track the low-frequency components, we apply a low-pass filter $\mathbf{B}$ on the density

distributions before measuring the difference between the guide and tracked distributions as

$$\boldsymbol{\rho}_{err} = \mathbf{B}\,\mathrm{adv}(\boldsymbol{\rho}_T^t, \boldsymbol{v}_T^t + \boldsymbol{u}) - \mathbf{B}\,\boldsymbol{\rho}_G^{t+1}, \tag{4.3.2}$$

where $\boldsymbol{\rho}_G^{t+1}$ is upsampled to the resolution of the tracked simulation using nearest-neighbor interpolation. In practice, $\mathbf{B}$ is a separable normalized Gaussian kernel. We estimate the correct size of this kernel to be 1.5× the size of the low-resolution grid cell. An illustration of the visual impact of the blur over the tracked simulation is found in Figure 4.3.2.

4.3.4. Objective Function and Derivative

In this section, we define the objective function to minimize as part of our smoke tracking problem, as well as its analytical derivative.

Our objective function is a weighted sum of a density tracking term ϕ_m, a regularization term ϕ_r, and a perturbation smoothness term ϕ_g, leading to an objective of the form:

$$\phi = k_m\phi_m + k_r\phi_r + k_g\phi_g. \tag{4.3.3}$$

As stated in Section 4.3.3, we minimize the blurred difference between the tracked and guide density distributions. We minimize the norm of this mismatch, giving the corresponding density matching objective

$$\phi_m = \frac{1}{2}\left\|\boldsymbol{\rho}_{err}\right\|^2 = \frac{1}{2}\left\|\mathbf{B}\,\mathrm{adv}(\boldsymbol{\rho}_T^t, \boldsymbol{v}_T^t + \boldsymbol{u}) - \mathbf{B}\,\boldsymbol{\rho}_G^{t+1}\right\|^2 \tag{4.3.4}$$

and the gradient

$$\frac{\partial\phi_m}{\partial\boldsymbol{u}}^T = \frac{\partial\mathrm{adv}(\boldsymbol{\rho}_T^t, \boldsymbol{v}_T^t + \boldsymbol{u})}{\partial\boldsymbol{u}}^T \mathbf{B}^T\,\boldsymbol{\rho}_{err}. \tag{4.3.5}$$

Since the use of a non-zero $\boldsymbol{u}$ makes the simulation deviate from a free, realistic smoke simulation, we want to foster solutions leading to a minimal modification of the velocity field. We thus add a regularization term to our objective, corresponding to the l^2-norm of $\boldsymbol{u}$:

$$\phi_r = \frac{1}{2}\left\|\boldsymbol{u}\right\|^2 \quad \text{with} \quad \frac{\partial\phi_r}{\partial\boldsymbol{u}}^T = \boldsymbol{u}. \tag{4.3.6}$$

Minimizing a linear combination of these two terms may lead to a highly spatially-varying solution. In order to increase the spatial smoothness of the vector field used to control the tracked simulation, we add a smoothness term to our objective, corresponding to the l^2-norm of the gradient of the velocity field modification, and given by

$$\phi_g = \frac{1}{2}\left\|\nabla\boldsymbol{u}\right\|^2 \quad \text{with} \quad \frac{\partial\phi_g}{\partial\boldsymbol{u}}^T = \nabla^T(\nabla\boldsymbol{u}). \tag{4.3.7}$$

A gradient-based optimization method is used to perform the solve. The gradient of the objective function can be trivially expressed in terms of the derivatives given by Equations (4.3.5), (4.3.6), and (4.3.7):

$$\frac{\partial \phi}{\partial \boldsymbol{u}} = k_m \frac{\partial \phi_m}{\partial \boldsymbol{u}} + k_r \frac{\partial \phi_r}{\partial \boldsymbol{u}} + k_g \frac{\partial \phi_g}{\partial \boldsymbol{u}}. \tag{4.3.8}$$

An illustration of the visual impact of the objective weights can be found in Figure 4.3.3.

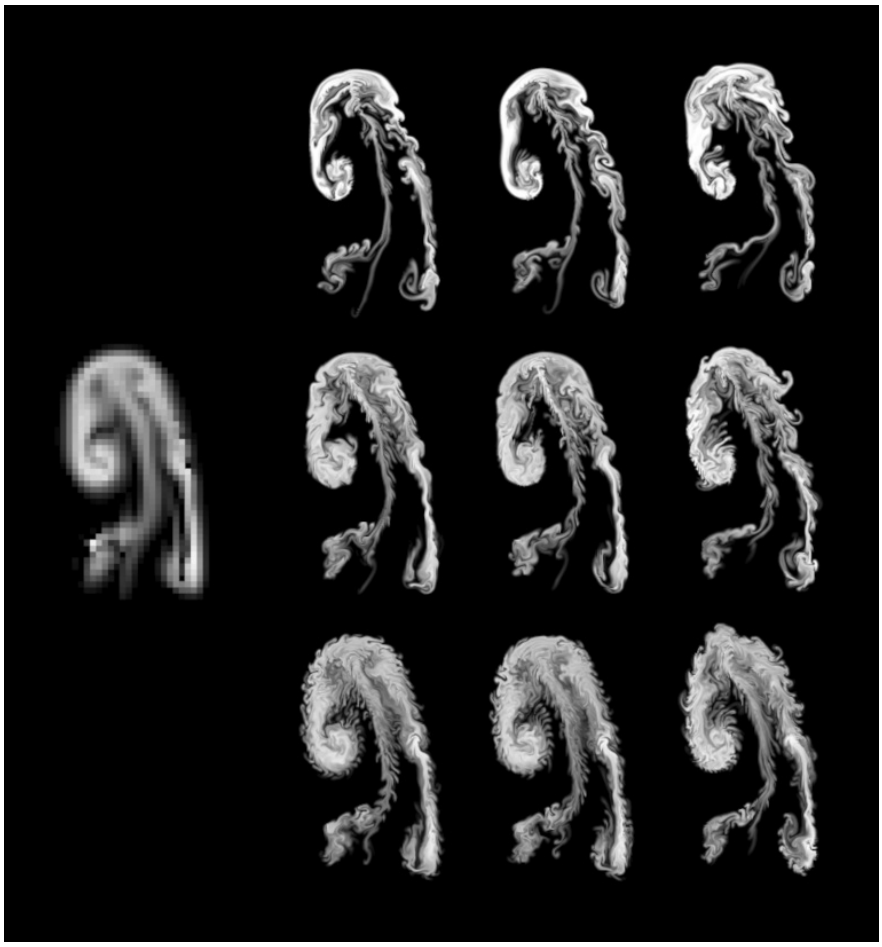


Fig. 4.3.3. Visual impact of the regularization weight k_r and the space-variation weight k_g. Left: Guide. Right: Tracked with $k_r \in \{0.0, 0.001, 0.03\}$ horizontally, and $k_g \in \{0.01, 0.1, 0.5\}$ vertically, in the usual order. Since the optimization is a proportion of three weights, we fixed arbitrarily one factor, here $k_m = 1.0$, in all our examples.

4.3.5. Advection Scheme Differentiability

To solve the optimization problem using a gradient-based method, we must compute the derivative of the advection operator adv(A,$\boldsymbol{v}$) with respect to the vector field $\boldsymbol{v}$, which implies that this operator has to be C^1-continuous in $\boldsymbol{v}$.

Although our method could be used with several types of advection schemes, we use a first-order semi-Lagrangian advection scheme. This choice is motivated by the simplicity to differentiate the first-order semi-Lagrangian advection scheme, its stability, and its low diffusivity compared to a purely Eulerian scheme. Linear interpolation has been used frequently when performing spatial density interpolation occurring in the semi-Lagrangian advection scheme. Unfortunately, the use of a linear interpolation does not lead to a differentiable advection operator. In order to enforce C^1-continuity of the advection operator, we use cubic Hermite splines to perform density interpolation. Tangents are enforced in each spatial dimension on both sides of a cell for the advected density field to be C^1-continuous. For each cell of the grid, an analytical expression of the density value obtained after advection can be expressed in terms of the position of the grid cell center (i,j), the velocity components $u[i,j]$, $u[i+1,j]$, $v[i,j]$, $v[i,j+1]$, the simulation time step Δt, and the density values stored in the $4 \times 4 = 16$ cells surrounding the position of the traced-back position of the center of (i,j) using cubic Hermite splines. We then take the derivative of this expression with respect to each velocity component, in order to apply the Jacobian matrix of the advection operator. Although we explained above for the 2D case for the sake of clarity, we can trivially extend this method in 3D, considering a third component for both positions and vectors, and using the $4 \times 4 \times 4 = 64$ cells surrounding the position of the traced-back position. Note that we discretize the different equations using a standard staggered MAC grid, storing velocities on cell faces, and both density and pressure at cell centers.

4.3.6. Divergence-free Constraint Enforcement

To satisfy the divergence-free constraint, we solve for a general velocity field $\tilde{\boldsymbol{u}}$ that has non-zero divergence, but we remove the divergence before adding the velocity perturbation to the simulation, leading to the equivalent optimization problem

$$\tilde{\boldsymbol{u}}^* = \arg\min_{\tilde{\boldsymbol{u}}} \phi(\mathbf{P}\tilde{\boldsymbol{u}}), \qquad \boldsymbol{u}^* = \mathbf{P}\tilde{\boldsymbol{u}}^*. \tag{4.3.9}$$

The gradient is then given by

$$\frac{\partial \phi}{\partial \tilde{\boldsymbol{u}}} = \mathbf{P}^T \frac{\partial \phi}{\partial \boldsymbol{u}}. \tag{4.3.10}$$

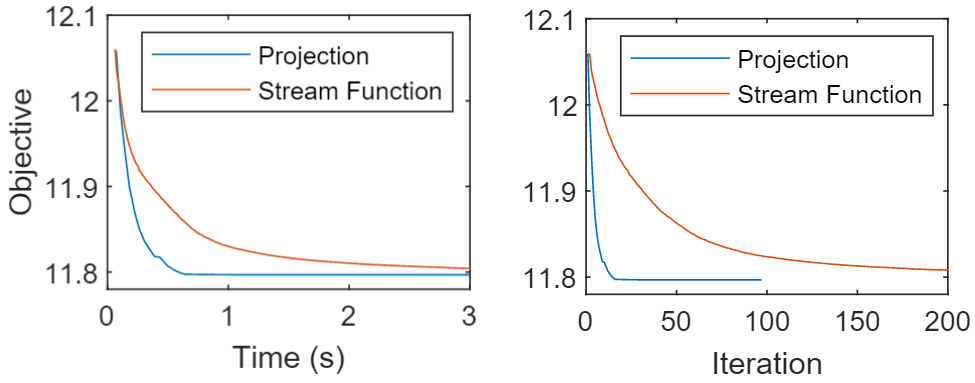


Fig. 4.3.4. Convergence of the optimization for one tracking step. We compare two different parameterizations of the divergence-free velocity field on the 100th time frame of the 2D example in Figure 4.4.3, starting with the same velocity and density fields.

An alternative would be to use a stream function [**2**] by defining $\boldsymbol{u} = \nabla \times \Psi$, with $\nabla\times$ being the curl operator; this velocity field is always divergence-free and thus does not require pressure projection. However, we found that this was not more efficient, as can be observed in Figure 4.3.4. Even though each iteration is cheaper, since no projection is necessary, many more iterations are required. We suspect that this is because changing exactly one velocity value u_i requires global changes in Ψ, as u_i is defined by the curl of Ψ, and thus discretized by differences of values of Ψ at different locations. Therefore the variables in the stream function parameterization are highly coupled, making it hard to optimize the objective.

4.3.7. Implementation

We solve the nonlinear optimization problem using L-BFGS. It increases memory consumption, but has faster convergence than a pure gradient-based method, and does not require the computation of the Hessian matrix, unlike Newton's method. L-BFGS stores a set of vectors to approximate the Hessian matrix, making it well suited for high-dimensional problems such as our case. We have not experienced unstable behaviors. Our method has been implemented as a *Mantaflow* [**123**] plugin. We use OpenMP for parallelization.

4.4. Results

We applied our smoke tracking method on several scenarios. All the simulations were run on a desktop PC with an i9-9900K 8-core CPU at 3.60 GHz with 128 GB of memory. In these examples, identical time steps were used on the low- and high-resolution simulations. We used $k_m = 1.0$, $k_r = 0.001$, $k_g = 0.1$, and a blur radius equivalent to $1.5\times$ the size of the

low-resolution grid cell. The set of weights has been found experimentally, corresponding to a good trade-off between the natural look of the high-resolution simulation and good density tracking with the guide. Those weights have been used to generate all the examples presented in this chapter.

4.4.1. Scenarios

2D Smoke Plume

As a first scenario, a plume in 2D, emitted from the bottom of the scene, spreads freely in the simulation domain. After computing a low-resolution guide simulation, we generated different simulations using our method with increasing upresolution factors, ranging from 2× to 16×. Figure 4.4.1 shows an image from each of these simulations, taken at an identical time. The high-resolution simulations generated using our method follow closely the guide regardless of the upresolution factor, while being enhanced with adapted high-frequency simulated details.

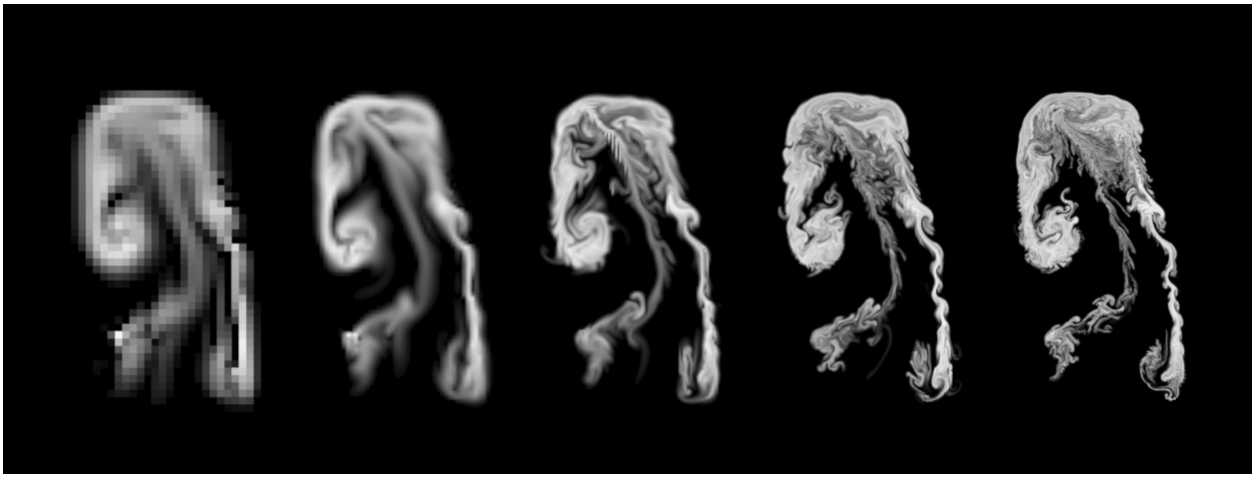


Fig. 4.4.1. 2D rising plume. Left to right: Guide (32×48), tracked 2×, 4×, 8×, and 16×. The larger the upresolution factor, the finer the added simulated details.

The blocky aspect of the density distribution of the guide is indicative of its coarse resolution. Recall that the low-resolution density is upsampled to the resolution of the tracked simulation using nearest neighbor interpolation when evaluating the density error.

Our 2D scenarios are included mainly to simplify the observation in a 2D image, and to better point out differences. 3D scenarios suffer from mixing smoke at different depths in a pixel, affected by absorption or occlusion. However, a 3D animation of smoke looks much more natural than its 2D counterpart, as can be appreciated in our images and animations.

3D Smoke Plume

Our method is trivially extended to 3D. In the next scenario, a smoke emitter placed in a pipe generates a rising plume in a 3D box domain. Buoyancy forces are applied. We performed three simulations. At first, a low-resolution ($32 \times 48 \times 32$) simulation, then two high-resolution ($128 \times 196 \times 128$) simulations: a free simulation and a simulation tracked with our method using the low-resolution simulation as a guide.

Figure 4.4.2 depicts an image from each of these simulations at an identical time. Even in this simple scenario, we can notice the different coarse appearances between the guide and free simulations. Indeed, the plume from the free simulation tends to rise faster, while the one from the guide appears slightly thicker. In contrast, the plume generated with our method in the tracked simulation exhibits the same coarse appearance than the one from the guide simulation, while being enhanced with high-frequency turbulent details.

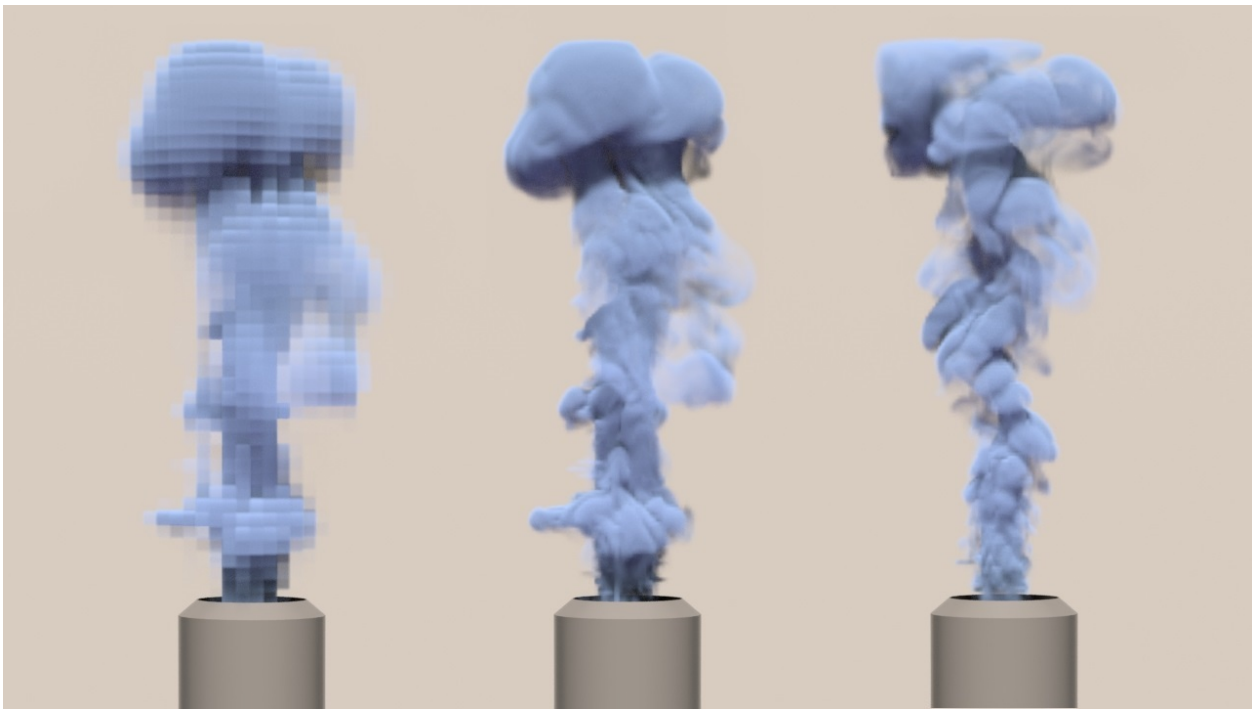


Fig. 4.4.2. 3D rising plume. Left to right: Guide, tracked, free.

Comparison with Previous Work

We compared our method with two previously discussed methods. First, we consider the tracking method of Nielsen et al. [**83**], which shares our general goals of control. As stated earlier, their method introduces a custom projection step, prescribing that the low-frequency

components of the simulated velocity field should be as close as possible to the input guide velocity field.

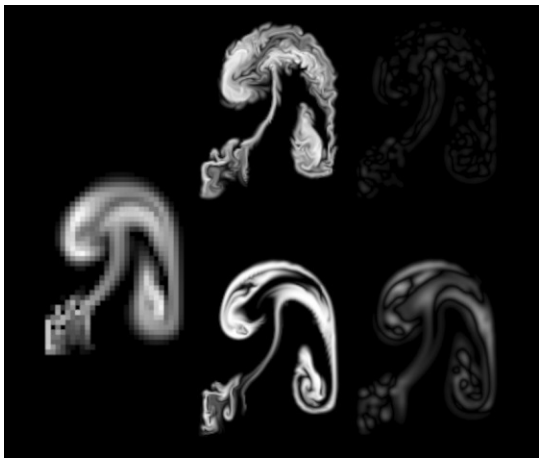


Fig. 4.4.3. 2D comparison with Nielsen et al. [**83**]. Left: Guide. Top center: Tracked using our method. Bottom center: Tracked using Nielsen et al. [**83**]. Right: RMS error for both methods.

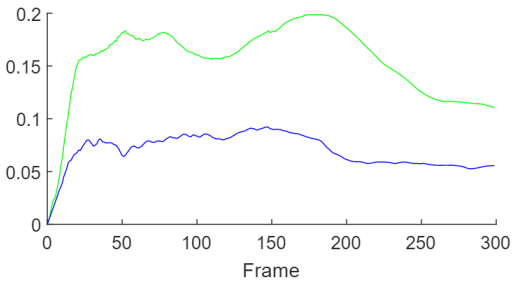


Fig. 4.4.4. RMS error per frame of the 2D simulations depicted in Figure 4.4.3. Blue: Our method. Green: Nielsen et al. [**83**].

In a simple 2D smoke plume scenario, we observe that while both our method and the one from Nielsen et al. [**83**] correctly track the coarse behavior of the smoke, the simulation tracked using our method turns out to be closer to the guide in terms of density distribution. Since the density distribution is the visible part of a smoke simulation, we found this error metric to be of interest. We support this observation by computing the RMS error between the guide and tracked blurred density fields. The blur operator is the same than the one used in our objective function, as it provides a reasonable approximation of a correct low-pass filter. Figure 4.4.3 shows an image from each of these simulations at an identical time frame, as well as the corresponding RMS error distribution. We summed these errors and divided by the number of occupied grid cells, before plotting them in the graph of Figure 4.4.4. It shows that the density distribution RMS error is roughly half for our method.

We also performed a 3D comparison of our method with the methods of Nielsen et al. [**83**] and Kim et al. [**60**] by tracking a 3D rising plume. We observe in Figure 4.4.5 that the density distribution of the tracked simulation generated using our method is visually closer to the guide. Again, note that 3D smoke density distribution appears more natural than their 2D counterparts. Faithfully representing density distribution of the guide density distribution is crucial in order to get a predictable simulation after the tracking step. In

the scene displayed in Figure 4.4.6, we cast shadows from spotlights, of the previous smoke simulations. Compared to the simulation generated using the method of Nielsen et al. [**83**], the emphasized shadows using our method are closer to the shadows generated by the guide. Working with density distributions from guide simulations could allow an artist to perform art-direction over a low-resolution density simulation in order to obtain targeted shadowing effects, being confident that the coarse aspect of the shadows generated by the tracked simulation would be similar.

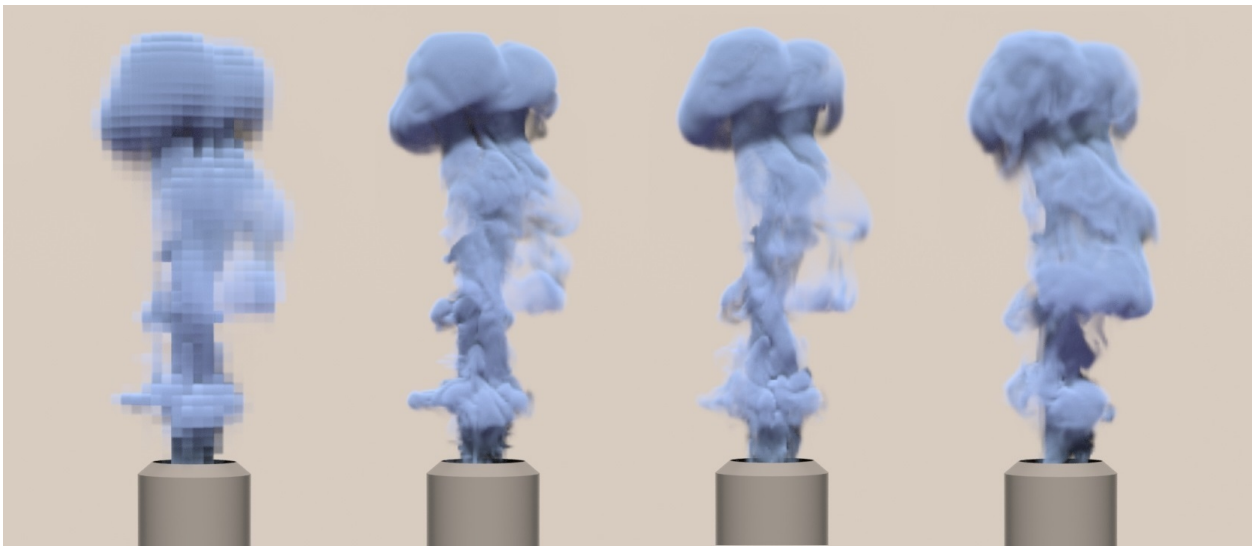


Fig. 4.4.5. Comparison of our method with different methods. From left to right: Guide, ours, Nielsen et al. [**83**], Kim et al. [**60**].

Compared to the method of Nielsen et al. [**83**], we explain differences in terms of high-resolution density distribution as follows. In the spatial interpolation occurring in the semi-Lagrangian advection, we are taking a weighted average of values from the previous time step. Averaging tends to smooth out sharp features, diffusing or dissipating them. The accuracy of the semi-Lagrangian advection depends on the resolution of the grid used to perform spatial interpolation. Indeed, the higher the resolution of the grid, the lower the diffusion. As a limitation to their method, Nielsen et al.. [**83**] themselves state that: "the amount of density-diffusion and -dissipation is significantly higher in low resolution. For this reason, features in the high-resolution guided simulation may deviate in intensity from the low-resolution simulation". For obvious reasons, our approach consisting in directly tracking the potentially highly-diffused low-resolution density field will result in a high-resolution density field more similar to the one from the guide.

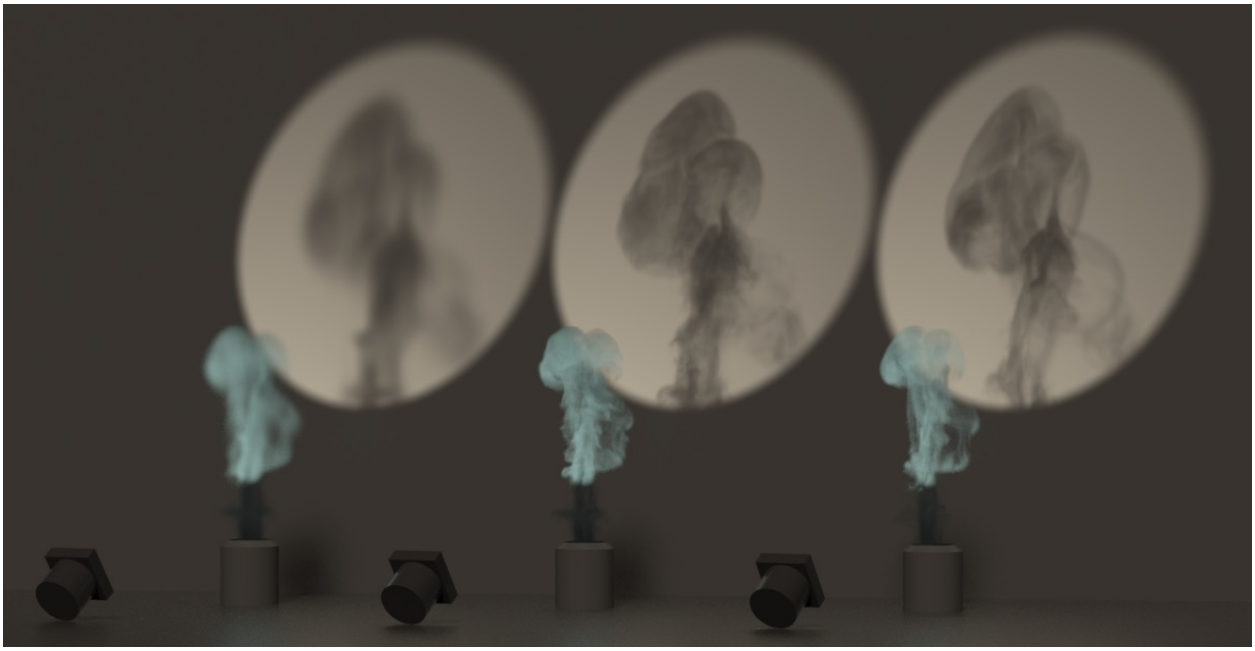


Fig. 4.4.6. 3D comparison under a different lighting. From left to right: Guide, ours, Nielsen et al. [**83**].

Smoke Interacting with Obstacles

Our method can also be used to track a guide simulation in which smoke interacts with objects. In the scenario depicted in Figure 4.4.7, a rising plume interacts with a static sphere. The tracked simulation, generated using our method and simulated without any obstacle in the scene, follows closely the motion of the guide smoke interacting with the obstacle, while giving birth to high-frequency plausible motion details. We can observe again how much the free simulation deviates from the guide simulation.

The fidelity to the guide simulation near boundaries is only achieved thanks to the density matching term, which is one component of our objective. However, tracking the guide density usually does not lead to exact solid boundary conditions. In some scenarios, one could prefer to accurately represent the frontiers between smoke and obstacles. In order to increase the spectrum of possibilities in terms of art direction, the use of boundary condition enforcement in our objective is left as a choice to the artist. This is illustrated in the scenario displayed in Figure 4.4.8, where a rising plume interacts with a set of three horizontal cylinders of different radii at different heights. Two tracked simulations have been generated. A first one in which the smoke is only tracked using the guide, and a second one in which the cylinders are added as boundaries in the tracked simulation. The shapes of the cylinders in the former

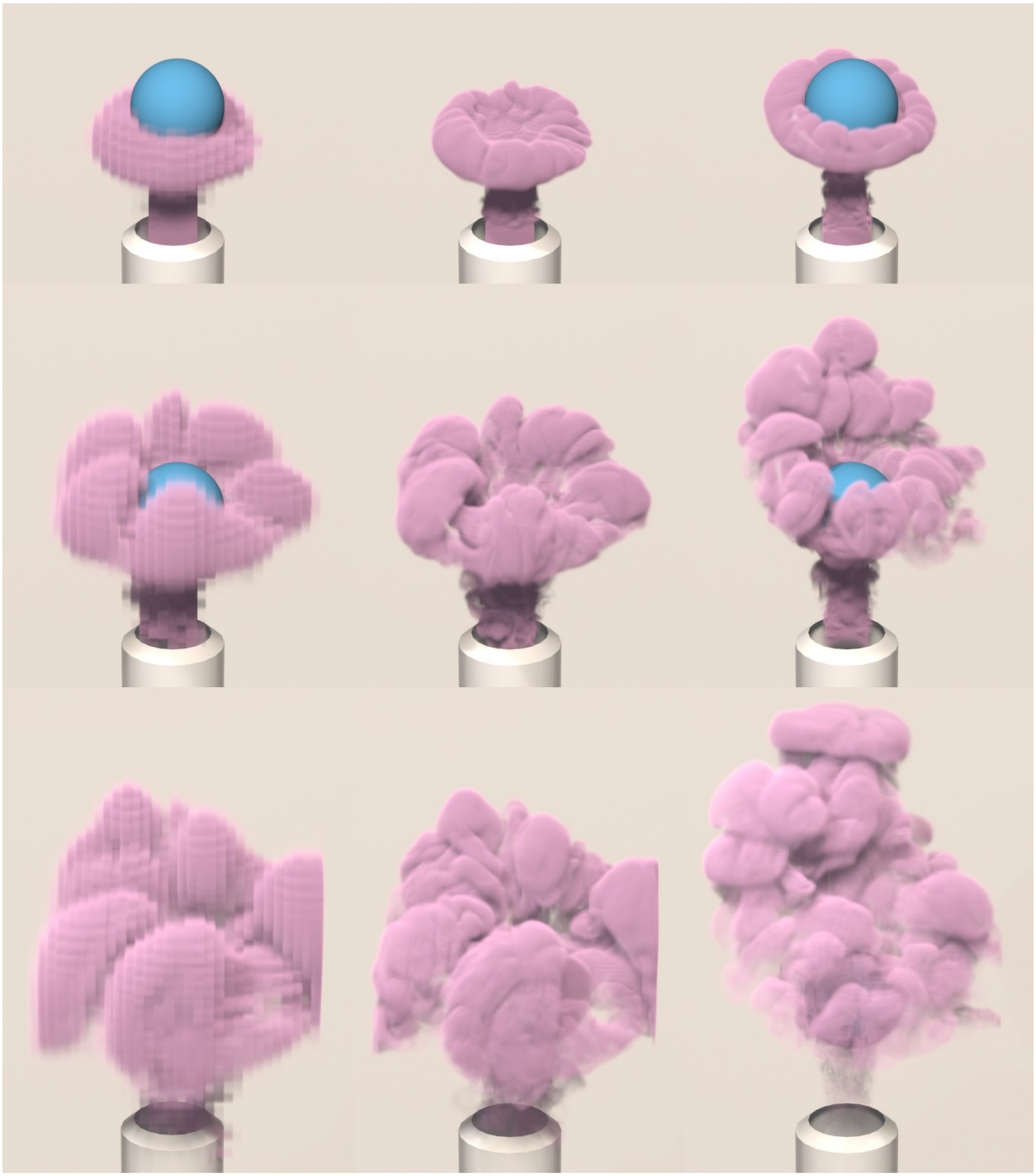


Fig. 4.4.7. Tracking smoke interacting with a sphere. Vertical: Images at three different times. Left: Guide with the sphere. Center: Tracked with our method without the sphere. Right: Free with the sphere.

simulation are still visible, even though their silhouettes are not as sharply delimited as with the actual cylinders used as obstacles.

Fig. 4.4.8. Tracking a low-resolution smoke interacting with cylinders (not displayed). Top left: Guide. Top right: Tracked simulations using our method without boundary conditions enforced. Bottom left: Tracked simulations using our method with boundary conditions enforced. Bottom right: Free with the cylinders, exhibiting a much different smoke distribution.

Fig. 4.4.9. Smoke interacting with an animated mesh. Left: A low-resolution smoke plume interacts with a dragon mesh moving horizontally, forming a smoke cushion underneath the dragon. Center: A high-resolution smoke plume generated with our method. Right: The corresponding high-resolution free simulation behaves much differently, quickly surrounding the dragon.

The obstacles do not have to remain static. In Figure 4.4.9, a dragon is moved horizontally to create a cushion of smoke underneath it. Our method follows well the guide, while a free animation at the same high resolution results in a much different distribution of the smoke.

Tracking Captured Data

We demonstrate the interest and robustness of our method by enriching captured data with physically simulated details. We used real-world reconstructed density distributions from Eckert et al. [**28**]. Since the original resolution ($100 \times 178 \times 100$) of these simulations is currently impracticable to be used as an input to our method, we generated a guide simulation by downsampling by half in each dimension the density distribution of a database simulation. We then computed a tracked simulation at a $200 \times 356 \times 200$ resolution using the generated guide. The resulting animation, thus generated in a grid twice as fine than the original data, closely follows the guide in both space and time, while being enriched with physically simulated small-scale details. In this example, faithfully reproducing the density is crucial since a deviation from the guide can be seen as a failure to respect the captured data.

Art-directed Guide

Since we only use a density distribution to guide a high-resolution simulation, our method is compatible with the use of non-physical and/or edited inputs. We demonstrated this concept in two scenarios.

First, we generated a non-physical scenario, depicted in Figure 4.4.11. In this scenario, we generated two plumes of smoke, modifying for each of them the orientation of the buoyancy forces to make them rise along the diagonal direction. We generated a guide by summing

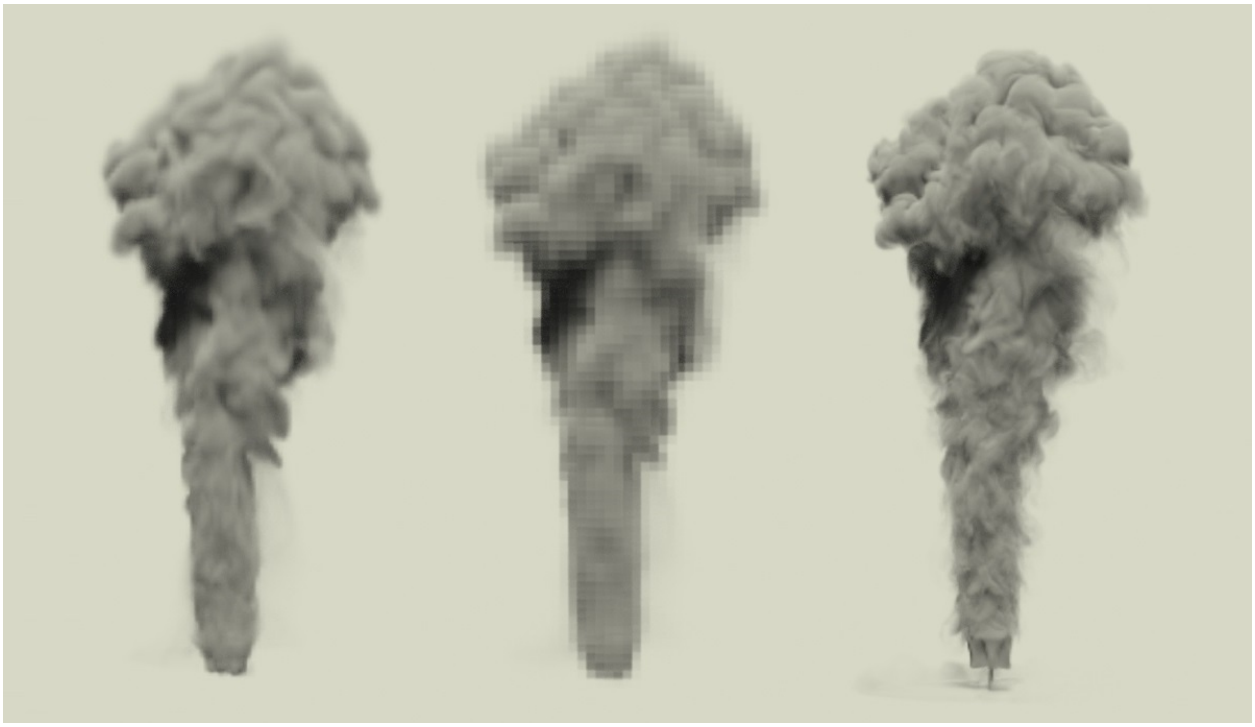


Fig. 4.4.10. Tracking real-world reconstructed captured data from Eckert et al. [**28**]. From left to right: Original data from Eckert et al. [**28**] ($100 \times 178 \times 100$), simulation obtained by downsampling these data ($50 \times 89 \times 50$), simulation generated the resulting simulation as a guide ($200 \times 356 \times 200$).

Fig. 4.4.11. Tracking the combination of two rising plumes. From left to right: A first low-resolution plume of smoke rising towards the top right corner, a second plume of smoke rising towards the top left corner, the result of the combination (by density summation) of the two previous simulations, and a tracked simulation generated using our method.

up the density values at each cell of the grid, before using the resulting data to guide a high-resolution simulation. Although the guide could not be physically simulated in a single simulation, our method is able to generate a plausible simulation, matching the guide. We

believe that the composition of a low-resolution guide with multiple animations, followed by the generation of a high-resolution physically-based simulation thanks to our tracking method, is an interesting direction to control fluids. In the field of image edition, artists are familiar with the generation of a targeted result by combining layers. For obvious reasons, this type of guiding could not have been achieved by tracking the sum of velocity fields using a method such as the one from Nielsen et al. [**83**].

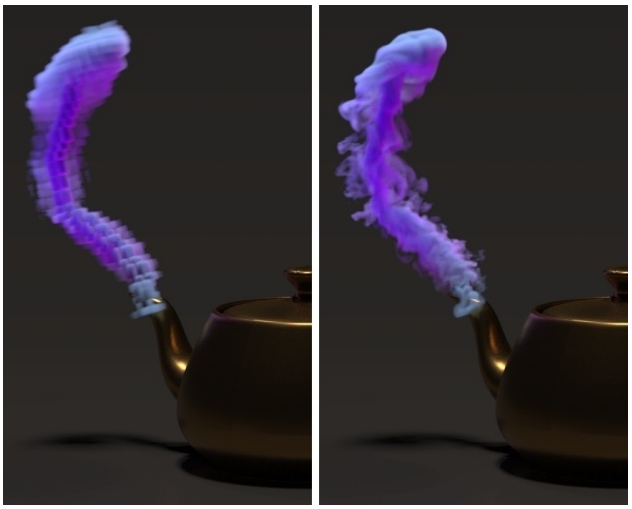


Fig. 4.4.12. Twisted smoke rising from a magic teapot. Left: A low-resolution rising plume is distorted to follow a spiral using a parametric function. Right: A high-resolution simulation, tracking the guide, is generated using our method.

In a second example, depicted in Figure 4.4.12, a thin plume of smoke is generated and distorted to follow a spiral when rising. The parametric function used to perform the distortion allows us to easily generate a targeted shape from a few parameters. A high-resolution smoke simulation is then generated using this non-physical simulation as a guide. Although the guide is highly aliased and non-physical, our method is able to generate a detailed smoke simulation that follows its general shape.

4.4.2. Statistics

Table 1 gives statistics about the simulations presented in this chapter. In these scenarios, high-resolution tracked and free simulations are 4× larger in every dimension than their respective low-resolution guide. An equivalent free simulation (same initial conditions and dimensions) takes 0.06s per frame in our 2D examples, and between 1.3s and 1.7s in our 3D examples; the associated tracked simulation takes 1.3s per frame in 2D, and between 60s to

Scene	Figure	Guide resolution	Time per frame		Factor	Memory (GB)
			Track	Free	Track/Free	
2D plume	Fig. 4.4.3	32×48	1.32s	0.06s	22.0×	0.05
3D plume	Fig. 4.4.2	$32 \times 48 \times 32$	76.6s	1.32s	58.0×	6.88
Sphere (without object)	Fig. 4.4.7	$32 \times 48 \times 32$	83.3s	1.42s	58.7×	6.88
Sphere (with object)	—		72.7s		51.2×	
Cylinders (without objects)	Fig. 4.4.8	$32 \times 48 \times 32$	63.1s	1.68s	37.6×	6.88
Cylinders (with objects)			61.6s		36.7×	
Dragon (with object)	Fig. 4.4.9	$32 \times 48 \times 32$	71.18s	1.83s	38.9×	6.88
Real-world data	Fig. 4.4.10	$50 \times 89 \times 50$	679.6s	6.84s	99.4×	31.9
Combined plumes	Fig. 4.4.11	$32 \times 48 \times 32$	80.3s	—	—	6.88
Magic teapot	Fig. 4.4.12	$32 \times 48 \times 32$	83.4s	—	—	6.88

Table 1. Statistics on our various upresolution examples. Note that in all these examples, the upresolution factor used is 4 in every dimension.

80s per frame in 3D. The reported memory usage is directly related to the size of the tracked simulation dimensions.

It is obvious that computing a free simulation, without any optimization, is much faster than a tracked simulation. Using an optimization scheme such as L-BFGS on large sets of variables comes with an important computational cost. From Table 1, we observe that our tracked simulations took between about 35–100× the computation times of equivalent free simulations in 3D. However, such free simulations do not follow any guide, and thus do not satisfy our goals.

We implemented the approach of Nielsen et al. [**83**] using the biconjugate gradient stabilized method (BiCGSTAB) with a diagonal preconditioner. A frame of the simulation depicted in Figure 4.4.5 took on average 123s to compute, compared to 76s for our method. Unfortunately, we did not implement the multi-grid solution described by Nielsen et al. [**83**], and so we cannot speculate about a fair full comparison with respect to computation times.

While there is room to improve our code, or optimizing with a proximal method such as ADMM could offer better performance, the increased computation factor due to our method should remain important given the size of the problem. For instance, our largest optimization contains close to 43 million values to optimize for. Fortunately, in the context of developing smoke edition tools and/or choosing between a number of simulations with different conditions, such a range of computational factors seems acceptable because it requires very little artist intervention to generate a final simulation which closely follows the guide.

In our tests and when the scenes where carefully designed (e.g., emitters located at the same position in space in both the low-resolution guide and the high-resolution simulation),

we did not observe any stability issue with our method. However, we observed slower convergence when the guide is further than physically plausible. The fact is that larger velocity modifications are required to make the high-resolution simulation match the guide.

4.5. Conclusions and Future Work

We have presented a method that uses a low-resolution simulation of smoke in order to guide a high-resolution simulation. We have demonstrated in a number of scenarios that the simulations generated using our method follow the global movements of the low-resolution simulation, while allowing the generation of fine details necessary to increase realism. By optimizing the tracking of density instead of a velocity field proximity, we observe that our simulations are visually and numerically closer to the guide, and their appearances in 3D are satisfying. Our method can thus be used to automatically and robustly increase the resolution of simulated, non-physical, and captured smoke animations. In combination with edition tools for smoke animations, we believe that working on lower-resolution 3D data is more affordable for artists. As such, our method offers great potential to shorten the iterative loop of artist-designed smoke animations, thanks to the simulation at a more efficient lower resolution.

Quantifying the quality of the matching is not trivial. For this purpose, we computed an RMS error between the blurred density distributions of the guide and the high-resolution simulation. However, more complex metrics (e.g., perception-based) might be more meaningful and should be used to this end. When using guides that are too far from a physically correct smoke simulation, larger velocity field modifications are required. In this situation, weights used in our objective function must be adjusted in order to reduce the amount of regularization. In cases where the density distributions of both high-resolution simulation and guide are too far from each other and do not overlap sufficiently, our tracking method fails to produce good results since this situation leads to a null gradient of the objective function. For this reason, scenes should be carefully designed, for example by placing emitters at the same position.

We have used an Eulerian discretization as the basis for our fluid tracking framework. In principle, our formulation could apply directly to Lagrangian approaches such as SPH when using a grid as intermediate representation. However, some applications might not allow for this memory-intensive detour, making a pure Lagrangian reformulation of our method another exciting direction for future work.

Chapter 5

Conclusion

5.1. Discussion

Although liquid control is a complex problem, it is crucial to provide artists with tools and techniques to efficiently generate targeted animations. Ideally, we would like these tools to allow animators to easily generate any type of art-directed animation featuring fluids. On the other hand, we want the controlled simulation to exhibit features specific to fluids, generated thanks to the use of physics-based solvers. Indeed, features such as splashes and ripples for liquids contribute to plausible aspects of the resulting animations, but they can prove challenging to artists to generate them with great realism. Sometimes, these two constraints can be contradictory, causing difficulties to propose a general solution to the liquid control problem. We believe that research should focus on the introduction of systems that combine efficient algorithms for the purpose of generating plausible and predictable animations in line with user inputs. These inputs should be ideally provided through the use of intuitive interfaces. In this thesis, we discussed two projects that can be used to help artists generate more predictable art-directed fluid animations.

We introduced an intuitive system to control particle-based liquid simulation using patches of precomputed animated liquids, stored in a database. In our system, each template instance can be temporally and spatially transformed using an intuitive graphical user interface. During the simulation, our system takes into account the resulting transformations to generate control forces and to manage temporary particles in order to correctly reproduce the template instances placed in the scene. We tested our system to generate several examples in various settings.

We also introduced an upresolution technique to generate a high-resolution smoke simulation that closely tracks density distribution of a low-resolution guide. At each simulation time step, we optimally modify the velocity field of the high-resolution simulation using a divergence-free perturbation in order to track the coarse density distribution of the guide.

The resulting high-resolution simulation is predictable since it closely follows the global aspect of the coarse guide density distribution, but is enhanced with fine simulated details. The technique, tested with various types of guides and scenes, generates plausible and predictable high-resolution smoke simulations.

Even though both systems produce satisfying results in general, extreme cases without fluid in an area or strong unnatural motions may not result in similar animations. This is because we preferred to remain more physically plausible than visually accurate. In fact, physical systems can react strongly to unnatural settings and affect many frames further in time.

5.2. Future Work

The two systems we presented can be used separately by an animator to produce art-directed fluid simulations. User-studies should be performed to measure the suitability of our tools (and our dedicated user interface for the system described in Chapter 3). In particular, interviewing FX artists would be of interest to validate the added value of these systems in a production context. For future projects, including FX artists early in the development process of such methods would definitely be beneficial and would ensure that the resulting tools do answer unfulfilled needs. In particular, it might impose different specifications, in line with real production contexts.

While various techniques have already been proposed to help artists produce art-directed fluids, many lines of research still have to be explored. We next discuss two of them closer to our heart.

New Types of User Interfaces

Sketch-based liquid control techniques in particular have been relatively little explored. Pan et al. [**86**] propose to control a liquid using three control metaphors including sketching strokes. Although inspiring, the resulting method has only been tested with very simple sketches, and the simplifications made to reduce its computation cost has lead to an approximate control.

Sketches are a natural way for many artists to express shapes and movements, and have proven to be convenient to describe and edit skeletal animations [**40**], as shown in Figure 5.2.1. Although the dynamics of a fluid is much more complex, investigating control methods based on sketching fluid streamlines would be of interest. For example, one could infer frequency components of a fluid velocity field from a set of sketched strokes. Longer strokes could locally define low-frequency components, while shorter strokes could be used

to control high-frequency details. Streamlines could also be hierarchically refined, allowing a control of the flow at different levels.

This strategy is already used in a hair grooming pipeline developed by Kaur et al. [**55**] and depicted in Figure 5.2.2. In this pipeline, hierarchical sculpting control curves are used to art-direct hair while preserving the hairstyle's structure. However, applying this idea to sketch a 3D fluid flow would not be trivial for a number of reasons.

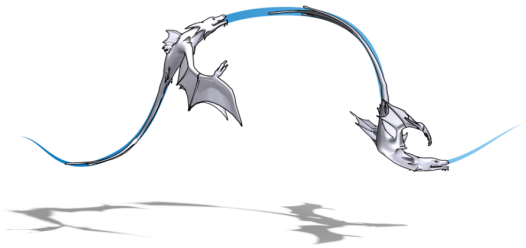


Fig. 5.2.1. Character animation sketched using the method of Guay et al. [**40**].

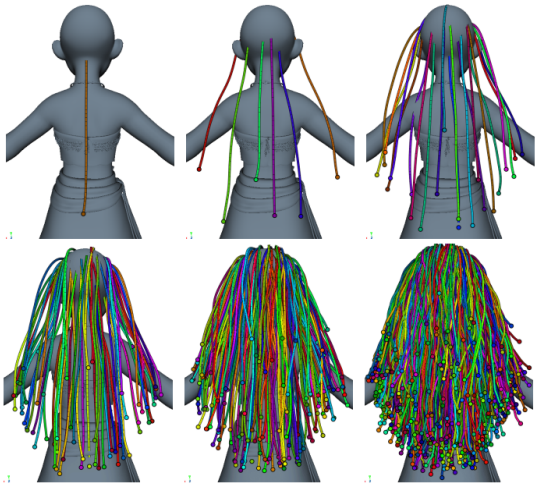


Fig. 5.2.2. Hierarchical control of hair using the pipeline from Kaur et al. [**55**].

First, hair strands are "attached" to control curves. In contrast, fluid is flowing along streamlines. Designing a tool to allow a user to edit a streamline and propagate the modification to its children in the hierarchy while maintaining the properties of the liquid (e.g., zero divergence of the velocity field) would be challenging.

Second, sketching in 3D is a well-known ill-posed problem. Since it is usually performed on a 2D screen, an infinite number of curves share the same projection on the screen. A solution is to impose extra constraints on the sketch, for example by considering that the 3D sketch lays on the surface of a 3D shape.

Third, a sketch must control a fluid in space and time. Sketch overlapping should be common, and a robust way of dealing with it should be found.

Overall, and even with the difficulties, this idea remains attractive and should be investigated in the future.

Other Types of Material

Slightly deviating from our initial fluid control problem, we do believe that investigating control methods for Material Point Method (MPM) simulations is of great interest. MPM can be seen as an extension of FLIP to solid-mechanics problems. In the past few years, this hybrid particle/grid method has been used to simulate a wide range of materials including snow and granular materials. In MPM, a local deformation is stored for each particle and updated at each simulation time step. Stress-based forces are defined and used to reduce the elastic potential energy of the system.

One could inject user control by applying a set of external forces on such a system. A force field could be easily defined, for example by describing both the material distribution and the target using signed distance functions (SDF), and by using a strategy similar to the one presented by Shi et al. [**112**]. However, the resulting control forces would be counteracted by internal stress-based forces, endangering the stability of the system.

Instead of using external forces, we could redefine the expression of the internal energy of the system in order for it to be minimal when the simulated material takes the form of a user-defined target shape. In this case, the system could rely on its internal stress-based forces to reach the target.

When applied to control a material such as snow, this type of approach would not be trivial to apply. Indeed, deformations applied on snow are mostly plastic. Formulating an elastic potential energy to force the simulated material to retrieve a user-defined target shape would very likely lead to an unnatural aspect of the snow. To tackle this issue, one could try to apply this technique on a portion of the simulated material, for example on a volumetric skeleton.

While non-trivial, we feel that the recent popularity of MPM will require adapted control methods.

5.3. Conclusion

Producing art-directed physics-based animations is a complex yet fascinating topic. Research has been conducted to provide control and increase the resolution of a fluid. However, much research is still necessary to give an artist the tools required to control fluids efficiently. In the future, it is not excluded that the rise of learning techniques could lead to new innovative fluid control techniques. We consider that the projects and ideas developed in this thesis are humble contributions to the field of fluid simulation control, and we look forward to what will follow next.

References

[1] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible SPH. *ACM Trans. Graph.*, 31(4), July 2012.

[2] Ryoichi Ando, Nils Thuerey, and Chris Wojtan. A stream function solver for liquid simulations. *ACM Trans. Graph.*, 34(4), July 2015.

[3] Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 87–96, 2005.

[4] Alexis Angelidis, Fabrice Neyret, Karan Singh, and Derek Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 25–32, 2006.

[5] Kai Bai, Wei Li, Mathieu Desbrun, and Xiaopei Liu. Dynamic upsampling of smoke through dictionary-based learning. *ACM Trans. Graph.*, 40(1), January 2020.

[6] Jernej Barbič, Marco da Silva, and Jovan Popović. Deformable object animation using reduced optimal control. *ACM Trans. Graph.*, 28(3), July 2009.

[7] Jernej Barbič, Funshing Sin, and Eitan Grinspun. Interactive editing of deformable simulations. *ACM Trans. Graph.*, 31(4), July 2012.

[8] Ronen Barzel, John R Hughes, and Daniel N Wood. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation'96*, pages 183–197. 1996.

[9] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 209–217, 2007.

[10] Jan Bender. SPlisHSPlasH, 2016. *https://github.com/InteractiveComputerGraphics/SPlisHSPlasH*.

[11] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 147–155, 2015.

[12] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. TRACKS: Toward Directable Thin Shells. *ACM Trans. Graph.*, 26(3):50:1–50:10, July 2007.

[13] Morten Bojsen-Hansen and Chris Wojtan. Generalized non-reflecting boundaries for fluid re-simulation. *ACM Trans. Graph.*, 35(4), July 2016.

[14] J.U. Brackbill and H.M. Ruppel. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65(2):314–343, 1986.

[15] Robert Bridson. *Fluid simulation for computer graphics*. 2015.

[16] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3), July 2007.

[17] Mathias Brousset, Emmanuelle Darles, Daniel Meneveaux, Pierre Poulin, and Benoît Crespin. Simulation and control of breaking waves using an external force model. *Computers & Graphics*, 57(C):102–111, June 2016.

[18] Gary Bruins and Jon Reisch. Rat-sized water effects in ratatouille. In *ACM SIGGRAPH 2007 Sketches*, page 69, 2007.

[19] Dong Joo Byun, James Mansfield, and Cesar Velazquez. Delicious looking ice cream effects with non-simulation approaches. In *ACM SIGGRAPH 2016 Talks*, pages 1–2, 2016.

[20] Deborah Carlson. Wave displacement effects for Surf's Up. In *ACM SIGGRAPH 2007 Sketches*, page 91, 2007.

[21] E. Catmull and F. Parke. A computer animated hand, 1972.

[22] Stephen Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 219–228, 2000.

[23] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. Graph.*, 36(4), July 2017.

[24] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 219–228, 2005.

[25] R. Clements and J. Musker. Moana, 2016. Walt Disney Animation Studios.

[26] Omar Cornut. Dear ImGui: Bloat-free immediate mode graphical user interface for C++ with minimal dependencies, 2020.

[27] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*, pages 61–76, 1996.

[28] Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. Scalarflow: A large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Trans. Graph.*, 38(6), November 2019.

[29] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, July 2002.

[30] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. *ACM Trans. Graph.*, 23(3):441–448, August 2004.

[31] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 15–22, 2001.

[32] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 2016.

[33] Zahra Forootaninia and Rahul Narain. Frequency-domain smoke guiding. *ACM Trans. Graph.*, 39(6), November 2020.

[34] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 23–30, 2001.

[35] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[36] Nick Foster and Dimitris Metaxas. Controlling fluid animation. In *Proceedings of the 1997 Conference on Computer Graphics International*, page 178, 1997.

[37] William Franklin Gates. *Animation of reactive fluids*. PhD thesis, University of British Columbia, 2002.

[38] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 139–148, 1997.

[39] James Gregson, Ivo Ihrke, Nils Thuerey, and Wolfgang Heidrich. From capture to simulation: connecting forward and inverse problems in fluids. *ACM Trans. Graph.*, 33(4):1–11, 2014.

[40] Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. Space-time sketching of character animation. *ACM Trans. Graph.*, 34(4), July 2015.

[41] Francis H Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics. Technical Report LADC-5288, U.S. Department of Energy, Office of Scientific and Technical Information, March 1962.

[42] Francis H Harlow and J Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[43] Xiaowei He, Ning Liu, Sheng Li, Hongan Wang, and Guoping Wang. Local poisson SPH for viscous incompressible fluids. *Computer Graphics Forum*, 31(6), September 2012.

[44] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 161–166, 2002.

[45] Jeong-Mo Hong and Chang-Hun Kim. Controlling fluid animation with geometric potential. *Comput. Animat. Virtual Worlds*, 15(3–4):147–157, July 2004.

[46] Ruoguan Huang, Zeki Melek, and John Keyser. Preview-based sampling for controlling gaseous simulations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–186, 2011.

[47] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner. Implicit incompressible SPH. *IEEE Transactions on Visualization & Computer Graphics*, 20(03):426–435, March 2014.

[48] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. SPH fluids in computer graphics. In *Eurographics 2014 - State of the Art Reports*, 2014.

[49] T. Inglis, Marie-Lena Eckert, J. Gregson, and N. Thürey. Primal-dual optimization for fluids. *Computer Graphics Forum*, 36(8):354–368, 2017.

[50] P. Jackson. The lord of the rings: The two towers, 2002. New Line Cinema and WingNut Films.

[51] S. Jeschke, C. Hafner, N. Chentanez, M. Macklin, M. Müller-Fischer, and C. Wojtan. Making procedural water waves boundary-aware. *Computer Graphics Forum*, 39(8):47–54, 2020.

[52] Stefan Jeschke, Tomáš Skřivan, Matthias Müller-Fischer, Nuttapong Chentanez, Miles Macklin, and Chris Wojtan. Water surface wavelets. *ACM Trans. Graph.*, 37(4), July 2018.

[53] Stefan Jeschke and Chris Wojtan. Water wave animation via wavefront parameter interpolation. *ACM Trans. Graph.*, 34(3), May 2015.

[54] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4), July 2015.

[55] Avneet Kaur, Maryann Simmons, and Brian Whited. Hierarchical controls for art-directed hair at disney. In *ACM SIGGRAPH 2018 Talks*, 2018.

[56] Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. Transport-based neural style transfer for smoke simulations. *ACM Trans. Graph.*, 38(6), November 2019.

[57] Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. Lagrangian neural style transfer for fluids. *ACM Trans. Graph.*, 39(4), July 2020.

[58] Doyub Kim, Oh-young Song, and Hyeong-Seok Ko. A semi-lagrangian cip fluid solver without dimensional splitting. *Computer Graphics Forum*, 27(2):467–475, 2008.

[59] Theodore Kim, Jerry Tessendorf, and Nils Thuerey. Closest point turbulence for liquid surfaces. *ACM Trans. Graph.*, 32(2):1–13, 2013.

[60] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):1–6, August 2008.

[61] Yootai Kim, Raghu Machiraju, and David Thompson. Path-based control of smoke simulations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 33–42, 2006.

[62] Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids. In *Eurographics Tutorial 2019*, 2019.

[63] Egor Larionov, Christopher Batty, and Robert Bridson. Variational stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Trans. Graph.*, 36(4), July 2017.

[64] John Lasseter. Principles of traditional animation applied to 3d computer animation. *Computer Graphics (SIGGRAPH Proceedings)*, 21(4):35–44, July 1987.

[65] Toon Lenaerts, Bart Adams, and Philip Dutré. Porous flow in particle-based fluid simulations. *ACM Trans. Graph.*, 27(3):1–8, August 2008.

[66] Siwang Li, Jin Huang, Fernando de Goes, Xiaogang Jin, Hujun Bao, and Mathieu Desbrun. Space-time editing of elastic motion through material optimization and reduction. *ACM Trans. Graph.*, 33(4), July 2014.

[67] Jia-Ming Lu, Xiao-Song Chen, Xiao Yan, Chen-Feng Li, Ming Lin, and Shi-Min Hu. A rigging-skinning scheme to control fluid simulation. *Computer Graphics Forum*, 38(7):501–512, 2019.

[68] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4), July 2013.

[69] Jamie Madill and David Mould. Target particle control of smoke simulation. In *Proceedings of Graphics Interface 2013*, pages 125–132, 2013.

[70] Pierre-Luc Manteaux, Ulysse Vimont, Chris Wojtan, Damien Rohmer, and Marie-Paule Cani. Space-time sculpting of liquid animation. In *Proceedings of the 9th International Conference on Motion in Games*, pages 61–71, 2016.

[71] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3), August 2004.

[72] Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. Surface turbulence for particle-based liquid simulations. *ACM Trans. Graph.*, 34(6), October 2015.

[73] Viorel Mihalef, Dimitris Metaxas, and Mark Sussman. Animation and control of breaking waves. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 315–324, 2004.

[74] Antoine Milliez, Robert W Sumner, Markus Gross, and Bernhard Thomaszewski. Haircontrol: a tracking solution for directable hair simulation. *Computer Graphics Forum*, 37(8):115–123, 2018.

[75] Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–18, 2008.

[76] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 1992.

[77] J.J. Monaghan. Simulating free surface flows with SPH. *J. Comput. Phys.*, 110(2):399–406, February 1994.

[78] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[79] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. In *Proceedings of ACM SIGGRAPH Asia 2008*, 2008.

[80] Rahul Narain, Jonas Zehnder, and Bernhard Thomaszewski. A second-order advection-reflection solver. *Proceedings of the ACM on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2(2), July 2019.

[81] Michael B Nielsen and Robert Bridson. Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph.*, 30(4):83:1–83:8, July 2011.

[82] Michael B Nielsen and Brian B Christensen. Improved variational guiding of smoke animations. *Computer Graphics Forum*, 29(2):705–712, 2010.

[83] Michael B Nielsen, Brian B Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 217–226, 2009.

[84] Michael B. Nielsen, Andreas Söderström, and Robert Bridson. Synthesizing waves from animated height fields. *ACM Trans. Graph.*, 32(1), February 2013.

[85] Michael B Nielsen, Konstantinos Stamatelos, Adrian Graham, Marcus Nordenstam, and Robert Bridson. Localized guided liquid simulations in bifrost. In *ACM SIGGRAPH 2017 Talks*, pages 1–2. 2017.

[86] Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. Interactive localized liquid motion editing. *ACM Trans. Graph.*, 32(6), November 2013.

[87] Zherong Pan and D. Manocha. Editing smoke animation using a deforming grid. *Computational Visual Media*, 3:369–378, 2017.

[88] Zherong Pan and Dinesh Manocha. Efficient solver for spacetime control of smoke. *ACM Trans. Graph.*, 36(5), July 2017.

[89] Mayur Patel and Noah Taylor. Simple divergence-free fields for artistic simulation. *Journal of graphics tools*, 10(4):49–60, 2005.

[90] Andreas Peer, Markus Ihmsen, Jens Cornelis, and Matthias Teschner. An implicit viscosity formulation for SPH fluids. *ACM Trans. Graph.*, 34(4), July 2015.

[91] Tobias Pfaff, Nils Thuerey, Jonathan Cohen, Sarah Tariq, and Markus Gross. Scalable fluid simulation using anisotropic turbulence particles. In *Proceedings of ACM SIGGRAPH Asia 2010*, 2010.

[92] Frédéric Pighin, Jonathan M. Cohen, and Maurya Shah. Modeling and editing flows using advected radial basis functions. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 223–232, 2004.

[93] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 209–217, 2000.

[94] Simon Premžoe, Tolga Tasdizen, James Bigler, Aaron Lefohn, and Ross T Whitaker. Particle-based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410, 2003.

[95] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 193–202, 2004.

[96] Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. Controlling liquids using meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 255–264, 2012.

[97] Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. Blending liquids. *ACM Trans. Graph.*, 33(4), July 2014.

[98] Bo Ren, Chenfeng Li, Xiao Yan, Ming C. Lin, Javier Bonet, and Shi-Min Hu. Multiple-fluid SPH simulation using a mixture model. *ACM Trans. Graph.*, 33(5), September 2014.

[99] Syuhei Sato, Yoshinori Dobashi, and Theodore Kim. Stream-guided smoke simulations. *ACM Trans. Graph.*, 40(4), July 2021.

[100] Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. Example-based turbulence style transfer. *ACM Trans. Graph.*, 37(4), July 2018.

[101] Syuhei Sato, Yoshinori Dobashi, and Tomoyuki Nishita. A combining method of fluid animations by interpolating flow fields. In *SIGGRAPH ASIA 2016 Technical Briefs*, 2016.

[102] Syuhei Sato, Yoshinori Dobashi, and Tomoyuki Nishita. Editing fluid animation using flow interpolation. *ACM Trans. Graph.*, 37(5), September 2018.

[103] Syuhei Sato, Yoshinori Dobashi, Yonghao Yue, Kei Iwasaki, and Tomoyuki Nishita. Incompressibility-preserving deformation for fluid flows using vector potentials. *The Visual Computer*, 31(6–8):959–965, June 2015.

[104] Syuhei Sato, Takuya Morita, Yoshinori Dobashi, and Tsuyoshi Yamamoto. A data-driven approach for synthesizing high-resolution animation of fire. In *Proceedings of the Digital Production Symposium*, DigiPro '12, pages 37–42, 2012.

[105] Arnaud Schoentgen, Pierre Poulin, Emmanuelle Darles, and Philippe Meseure. Particle-based liquid control using animation templates. *Computer Graphics Forum*, 39(8), November 2020.

[106] Arnaud Schoentgen, Jonas Zehnder, Pierre Poulin, Bernhard Thomaszewski, Philippe Meseure, and Emmanuelle Darles. A density-accurate tracking solution for smoke upresolution. *The Visual Computer*, 36:2299–2311, July 2020.

[107] Joshua Schpok, William Dwyer, and David S. Ebert. Modeling and animating gases with simulation features. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 97–105, 2005.

[108] Christian Schulz, Christoph von Tycowicz, Hans-Peter Seidel, and Klaus Hildebrandt. Animating deformable objects using sparse spacetime constraints. *ACM Trans. Graph.*, 33(4), July 2014.

[109] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2):350–371, 2008.

[110] Songdong Shao and Edmond Lo. Incompressible SPH method for simulating newtonian and non-newtonian fows with a free surface. *Advances in Water Resources*, 26(7):787–800, July 2003.

[111] Lin Shi and Yizhou Yu. Controllable smoke animation with guiding objects. *ACM Trans. Graph.*, 24(1):140–164, January 2005.

[112] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 229–236, 2005.

[113] Seung-Ho Shin and Chang-Hun Kim. Target-driven liquid animation with interfacial discontinuities. *Comput. Animat. Virtual Worlds*, 18(4–5):447–453, September 2007.

[114] Barbara Solenthaler. Predictive-corrective incompressible SPH. *ACM Trans. Graph.*, 28(3), August 2009.

[115] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 121–128, 1999.

[116] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 129–136, 1995.

[117] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Trans. Graph.*, 32(4), July 2013.

[118] Alexey Stomakhin and Andrew Selle. Fluxed animated boundary method. *ACM Trans. Graph.*, 36(4), July 2017.

[119] Tuur Stuyck and Philip Dutré. Sculpting fluids: A new and intuitive approach to art-directable fluids. In *ACM SIGGRAPH 2016 Posters*, 2016.

[120] Andre Pradhana Tampubolon, Theodore Gast, Gergely Klár, Chuyuan Fu, Joseph Teran, Chenfanfu Jiang, and Ken Museth. Multi-species simulation of porous sand and water mixtures. *ACM Trans. Graph.*, 36(4), July 2017.

[121] Jerry Tessendorf. Simulating ocean water. *SIGGRAPH Course Notes*, 2002.

[122] Nils Thuerey. Interpolations of smoke and liquid simulations. *ACM Trans. Graph.*, 36(1), February 2017.

[123] Nils Thuerey and Tobias Pfaff. MantaFlow, 2018. *http://mantaflow.com*.

[124] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. Detail-preserving fluid control. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 7–12, 2006.

[125] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22(3), July 2003.

[126] Christopher D. Twigg and Doug L. James. Many-worlds browsing for control of multibody dynamics. *ACM Trans. Graph.*, 26(3):14–21, July 2007.

[127] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. *Computer Graphics Forum*, 37(8):171–182, 2018.

[128] Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. A multi-pass GAN for fluid flow super-resolution. *Proceedings of the ACM on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2(2), July 2019.

[129] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics (SIGGRAPH Proceedings)*, 22(4):159–168, August 1988.

[130] Chris Wojtan, Peter J. Mucha, and Greg Turk. Keyframe control of complex particle systems using the adjoint method. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 15–23, 2006.

[131] Magnus Wrenninge and Doug Roble. Fluid simulation interaction techniques. In *ACM SIGGRAPH 2003 Sketches & Applications*, page 1, 2003.

[132] Xiangyun Xiao, Hui Wang, and Xubo Yang. A CNN-based flow correction method for fast preview. February 2019.

[133] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. TempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Trans. Graph.*, 37(4), July 2018.

[134] Guowei Yan, Zhili Chen, Jimei Yang, and Huamin Wang. Interactive liquid splash modeling by user sketches. *ACM Trans. Graph.*, 39(6), November 2020.

[135] Ben Yang, Youquan Liu, Lihua You, and Xiaogang Jin. A unified smoke control method based on signed distance field. *Computers & Graphics*, 37(7):775–786, November 2013.

[136] Shuqi Yang, Shiying Xiong, Yaorui Zhang, Fan Feng, Jinyuan Liu, and Bo Zhu. Clebsch gauge fluid. *ACM Trans. Graph.*, 40(4), July 2021.

[137] Zhi Yuan, Fan Chen, and Ye Zhao. Pattern-guided smoke animation with lagrangian coherent structure. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, 2011.

[138] Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski. An advection-reflection solver for detail-preserving fluid simulation. *ACM Trans. Graph.*, 37(4), July 2018.

[139] Guijuan Zhang, Dengming Zhu, Xianjie Qiu, and Zhaoqi Wang. Skeleton-based control of fluid animation. *The Visual Computer*, 27:199–210, 2010.

[140] Shuai Zhang, Xubo Yang, Ziqi Wu, and Haibo Liu. Position-based fluid control. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, pages 61–68, 2015.

[141] Xinxin Zhang, Robert Bridson, and Chen Greif. Restoring the missing vorticity in advection-projection fluid solvers. *ACM Trans. Graph.*, 34(4), July 2015.

[142] Ye Zhao, Zhi Yuan, and Fan Chen. Enhancing fluid animation with adaptive, controllable and intermittent turbulence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 75–84, 2010.

[143] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3), July 2005.