Dimension reduction for fluid simulation and animation

Chloé Paliard

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAT023

Thèse de doctorat

Dimension reduction for fluid simulation and animation

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique, données, IA

Thèse présentée et soutenue à Palaiseau, le 27/06/2024, par

CHLOÉ PALIARD

Composition du Jury :

Mathieu Desbrun Advanced Researcher, INRIA/Ecole Polytechnique, France	Président/Examinateur
Florence Bertails-Descoubes Directrice de Recherche, INRIA Grenoble Rhône-Alpes, France	Rapportrice
Maud Marchal Professeure, Université/INSA de Rennes, France	Rapportrice
Barbara Solenthaler Professeure, ETH Zurich, Suisse	Examinatrice
Marco Cagnazzo Professeur, Université de Padoue, Italie	Directeur de thèse
Kiwon Um Maître de Conférences, Télécom Paris, France	Co-encadrant de thèse
Nils Thuerey Professeur, Université Technique de Munich, Allemagne	Co-encadrant de thèse
Jean-Marc Thiery Senior Research Scientist, Adobe, France	Invité

Remerciements

I want to start by thanking the members of my jury. Barbara, Florence, Mathieu, Maud, thank you for accepting to be part of my PhD jury. Thank you for your relevant and interesting remarks and questions, I really appreciated our discussions during the defense, and thank you for your very kind feedback on my manuscript and presentation. More specifically, thank you to my reviewers, Florence and Maud, for thoroughly reading my manuscript and for your very detailed reports. I also want to thank the members of my CSI, Barbara and Florence, for giving me great scientific and personal advice all along my PhD. Florence, merci pour tes nombreux conseils du premier CSI à ma soutenance, et merci de m'avoir montré que la recherche pouvait être faite d'une manière différente. Merci de t'être rendue aussi disponible, ton aide m'a été très précieuse.

I want to thank my PhD advisors, Kiwon, Nils and Marco. Thank you for accepting to supervise and advise me during my PhD, and for your trust. Marco, thank you for accepting to join us on this thesis subject and for supervising me as well as you could from Italy. Nils, thank you for your presence during my first project and for your very relevant and sharp scientific advice. Kiwon, thank you for your strong presence at the beginning of my PhD, and for trying your best although a lot of (positive) change happened in your life during my thesis.

Je souhaite désormais remercier l'équipe IMAGES. J'ai d'abord été accueillie dans le CG group de Tamy et Jean-Marc, composé également de tou.tes les ancien.nes membres avec qui nous avons partagé de nombreux verres à la Butte aux Cailles et qui n'ont jamais manqué de conseils pour me guider au début de ma thèse. Je voudrais surtout remercier mes co-thésard.es pour deux ans : Jérémie, Tong, avec une mention très spéciale pour Elie et Alban.

Ensuite j'ai intégré la bien plus grande famille qu'est l'équipe IMAGES, avec ses permanent.es et doctorant.es d'une qualité humaine rare. Merci infiniment à tou.tes les permanent.es pour votre présence, vos conseils, votre humour et votre bienveillance. Plus particulièrement, j'aimerais remercier chaleureusement Florence et Yann. Merci à vous deux de m'avoir aidée à une période critique et de vous être rendu.es disponibles alors que vos emplois du temps étaient bien remplis. Merci également à Yann et aux différent.es membres du conseil des doctorant.es (Nicolas, Emanuele, Mateus, Alban, Rebeca, Zoé, Gwilherm) pour l'attention immense portée au bien-être de tous et toutes, et pour l'investissement souvent invisible.

Evidemment un merci immense à tou.tes les doctorant.es de l'équipe, avec qui j'ai partagé mes journées à Saclay aussi bien que les restos thaï ou les verres au Diamant, ainsi que les soirées endiablées et inoubliables au Divan du monde. Plus particulièrement, j'ai la chance d'avoir de véritables ami.es dans cette équipe, et sans vous je n'aurais certainement pas soutenu ma thèse. Merci Rebeca, Raph (Remé), Marie, Pierrick, Robin, Antoine, Erwan, Nico et Raph. Merci également à Inès, Matthis, Emma, Zoé et Gwilherm. J'espère qu'on se refera un Parc Astérix très bientôt !

J'aimerais également remercier Delphine, secrétaire du département IDS, pour son aide précieuse de mon premier jour comme doctorante jusqu'à ma soutenance. Nous avons

beaucoup de chance de t'avoir, ta gentillesse et ta disponibilité ainsi que ton efficacité sont indispensables au bon fonctionnement du département.

Ensuite, j'aimerais remercier tou.tes les membres d'Adobe Paris, permanent.es et stagiaires, qui m'ont d'abord accueillie en stage en 2022 puis comme visiteuse régulière cette dernière année. Merci à Tamy et son formidable Paris lab pour votre confiance. En particulier, je voudrais remercier ma tutrice et mon tuteur de stage de césure, Rosalie et Arthur. Vos mots toujours encourageants, votre reconnaissance et votre confiance m'ont donné une force immense pour la deuxième partie de ma thèse, merci infiniment. Arthur, j'ai une chance folle d'avoir pu travailler avec toi quotidiennement. Ton intérêt pour divers sujets scientifiques mais surtout ton humanité et ta sensibilité, ainsi que ton humour, ont été et sont toujours très précieux pour moi. Merci.

J'en viens désormais à mes collègues de quatrième année de thèse du Adobe Paris lab : Axel, Elie, JM, Jose et Loïs. J'aimerais d'abord souligner que vous êtes des chercheurs d'une qualité rare, et je suis honorée d'avoir eu votre confiance et d'avoir partagé vos réflexions scientifiques. JM et Elie, merci de m'avoir proposé de travailler avec vous pour la seconde partie de ma thèse, votre confiance a été très précieuse pour moi. Jose, thank you for joining this project with us, it was a pleasure to work with a great researcher like you. Thank you for your kindness and your advice. Elie, tu as toujours été présent et volontaire pour m'aider, du premier au dernier jour de ma thèse, et tu es par ailleurs un ami précieux, merci. Loïs, merci pour ton engagement sur le projet, mais surtout merci pour les discussions toujours passionnantes sur divers sujets de société. Tes blagues fines et toujours bien placées étaient un élément indispensable au bon déroulement de nos réunions. Merci aussi pour ton amitié. JM, l'enthousiasme et l'optimisme incarnés, tu m'impressionneras toujours. Merci d'avoir cru en moi et de t'être autant engagé dans ma thèse, tu as été un quatrième encadrant hors paire. Désolée de ne pas avoir prolongé encore ma thèse, j'aurais travaillé sur les PMVC et les peintures dendritiques avec toi avec grand plaisir ! Enfin Axel, je crois qu'aucun mot ne saurait décrire la gratitude que je ressens envers toi. J'ai rarement vu une personne aussi investie dans ses projets, tu y mets tout ton coeur en plus de ton cerveau, tu es un humain et surtout un ami formidable. Merci pour ta disponibilité, ton intelligence, ta bienveillance, ta gentillesse et ton humour. Mais aussi merci pour les potins et même pour les petits pics quand on était tous les deux à bout en deadline SIGGRAPH, alors que Loïs était très détendu et que JM enchainaît les canettes de Coca... Cette scène restera gravée à jamais dans ma mémoire.

Je voudrais désormais remercier ma famille. Maé merci de m'avoir accueillie ces deux dernières semaines dans l'hôtel de la rue Mayet, ça m'a permis d'aborder ma soutenance sereinement. Merci également pour ta présence indispensable depuis toujours. Loïc, Emilie et Neris, merci pour votre soutien et de vous être toujours intéressé.es à ce que je faisais, même si ça vous semblait toujours un peu étranger. Benji, merci pour ton soutien infaillible dans la thèse comme dans le reste. Merci pour ton humour plus ou moins fin, ta présence et ton écoute. Mélino, merci pour ton soutien, ton écoute et tes mots, je suis heureuse de te compter parmi mes amies. Enfin, Papa et Maman, merci d'avoir toujours cru en moi et de m'avoir encouragée tout en me laissant une grande indépendance. Merci à vous deux de m'avoir transmis votre bienveillance et votre gentillesse, ainsi que votre résilience. Papa, merci de m'avoir transmis la persévérance, le dépassement de soi et l'intégrité et la droiture comme valeurs non négociables. Maman, merci de m'avoir transmis l'attention, l'écoute, la sensibilité au monde et aux gens. Mais également merci de m'avoir appris à ne pas me laisser faire et à m'affirmer, ça m'a été indispensable pour terminer cette thèse.

Je voudrais maintenant remercier Géraldine, ma soeur. Tu n'as absolument jamais cessé de croire en moi, tu m'appelais ton petit génie ou encore Einstein au lycée et tu faisais des plans pour le jour où j'aurais mon propre théorème et où tu pourrais arrêter de travailler car

j'aurais assez de succès pour nous faire vivre toutes les deux convenablement. On a tout vécu toutes les deux, une amitié comme la notre c'est rare et très précieux, et c'est une grande force pour traverser des épreuves telles qu'une thèse. Merci pour tout.

Je pense aussi particulièrement à ma deuxième maman. Tu as toujours cru en moi et me l'as toujours dit et montré. Je suis sûre que tu aurais été très fière de me voir soutenir et devenir docteure, et y penser me fait chaud au coeur. Je suis heureuse d'emprunter aujourd'hui la voie de la danse qui était la tienne.

Ensuite, j'aimerais justement remercier les amies et copines que j'ai connues par la danse : Zoé, Mathilde, Floriane, Margot et Charlotte. Merci pour votre soutien et vos mots pendant les derniers mois de cette thèse. Je tiens également à remercier mes deux professeures de danse de cette année, Juliette et Lucie, qui m'ont beaucoup aidée à prendre confiance en moi.

Enfin, merci Théo de toujours être là, du moins quand les trains te le permettent... Merci Anaïs, Madeleine, PF, Raphaëlle et Margaux pour votre amitié et votre soutien. Merci Alex, Clémence et Ananas pour vos mots, votre écoute et votre soutien sans faille, ainsi que pour tous les bons moments partagés ensemble. Merci aux relecteur.ices de mon manuscrit, qui m'ont aussi fait répéter ma soutenance et sont surtout des ami.es formidables : Octave, Axel, Rebeca, Elie. Merci Pablo, Mathu, Julo et Rekkos de me donner un espace amical où je me sens à ma place et aimée, et merci pour votre folie. Merci Pablo pour ton amitié précieuse. Merci Laura pour ta présence, ton soutien et ton écoute. Merci d'être toujours là dans les moments importants et surtout dans les moments galères ou difficiles. Merci Renaud et Chloé pour votre amitié, qui est indispensable à mon bonheur depuis bientôt dix ans.

Soso, mon soleil, merci pour ta présence et ton soutien quotidiens depuis la fin de la première année à Télécom. Merci aussi pour ton ordi, qui a été mon plus fidèle compagnon ces derniers mois. Mais surtout merci de toujours voir et me montrer le bon côté des choses, merci pour ta force, pour ton écoute et pour tes mots, ainsi que pour les rires.

Rebeca, Dr Vetil, merci pour ta présence plus que quotidienne, pour ta confiance, ton écoute et tes mots. Merci pour les très nombreux fous rires et merci aussi pour les larmes. On s'est trouvées il y a un an et demi quand nos thèses nous avaient poussées à bout, et je ressens une fierté immense en nous voyant être arrivées là où on est aujourd'hui, toutes les deux docteures. Merci pour tout.

J'aimerais écrire quelques mots supplémentaires pour remercier Sylvie Coussot, psychologue pour les doctorant.es de l'IPP. Son aide m'a été précieuse, que ce soit pour ma thèse ou mon avenir ou bien sûr ma santé mentale en général. Une grande majorité de doctorant.es présentent des symptômes dépressifs pendant leur thèse, et c'est loin d'être normal. N'hésitez pas à consulter un.e professionnel.le de santé si vous êtes doctorant.e et que vous n'allez pas bien, même si vous pensez que "ça n'est pas grand chose". De même, n'hésitez pas à aiguiller vos étudiant.es si vous êtes encadrant.e.

Résumé

L'informatique graphique est un domaine en pleine expansion, qui vise principalement à produire du contenu visuel sur des supports numériques, pour des applications telles que les effets visuels, les jeux vidéo ou la conception assistée par ordinateur. Un des objectifs de la recherche en informatique graphique est d'améliorer le photoréalisme des objets et des matériaux virtuels, ou d'animer des personnages et des phénomènes naturels de la manière la plus réaliste possible. Néanmoins, malgré les améliorations considérables des performances du matériel graphique ainsi que des avancées algorithmiques majeures au cours du temps, certains phénomènes naturels restent extrêmement coûteux à simuler. Par exemple, capturer la dynamique et les comportements complexes d'un fluide nécessite une quantité importante de mémoire et de calculs. Cependant, les ressources sont limitées et la consommation d'énergie est donc coûteuse en termes d'argent et, surtout, a un impact important sur l'environnement et les vies humaines. Pour ces différentes raisons, de nombreuses recherches se sont concentrées sur l'optimisation de ces simulations.

Plusieurs pistes ont été proposées et explorées pour améliorer les performances des simulations de fluides, qui sont animées par la résolution d'équations différentielles partielles (EDP). Ces équations peuvent être extrêmement coûteuses à résoudre, en particulier dans le cas des fluides où les EDP en question sont les équations de Navier-Stokes, fortement non linéaires. Par conséquent, il reste particulièrement difficile de produire des écoulements de fluides en temps réel à des résolutions élevées. Un sujet important, qui s'est développé à un rythme sans précédent dans les années 2000, est l'utilisation de réseaux de neurones profonds en conjonction avec des solveurs basés sur la physique afin de réduire le temps de calcul pour la résolution de ces EDP. Cette approche basée sur les données s'est avérée efficace pour imiter de tels solveurs (ou des parties de ceux-ci), par exemple pour recréer de très petits détails tels que des gouttelettes. Dans cette thèse, nous explorons d'abord l'utilisation de l'apprentissage profond pour créer un espace réduit dans lequel un solveur peut opérer à moindre coût, tout en produisant des solutions de haute qualité. En effet, la plupart des travaux antérieurs axés sur la réduction de dimension utilisent un échantillonnage linéaire traditionnel pour réduire les degrés de liberté d'une simulation, et nous proposons d'autres espaces plus pertinents, sans contrainte prédéfinie. Nous proposons ainsi un modèle qui permet de simuler des écoulements turbulents à une résolution quatre fois supérieure à celle de l'entrée dans chaque dimension, avec des performances d'exécution améliorées par rapport à un solveur haute résolution. Ce modèle est présenté pour divers scénarios physiques et comparé à des techniques traditionnelles.

En plus d'utiliser des modèles d'apprentissage profond pour réduire les degrés de liberté d'une simulation, nous proposons de relier les communautés de la simulation basée sur la physique et du traitement de la géométrie, afin d'améliorer les performances de l'animation

de fluides. En géométrie, il est de plus en plus courant d'utiliser des opérateurs intrinsèques – qui ne nécessitent pas de représentation 3D de la surface – car ils peuvent être plus robustes que les opérateurs extrinsèques. D'autre part, un défi récurrent dans la simulation de fluides est la gestion des interactions fluide-solide. En particulier, certains scénarios d'application nécessitent de résoudre les équations de Navier-Stokes principalement sur la surface, alors que certains modèles simulent l'ensemble du domaine 3D, ce qui est extrêmement coûteux. Partant de ces observations, nous souhaitons utiliser les contributions récentes sur les opérateurs intrinsèques pour simuler des fluides sur des surfaces 3D avec des coûts réduits. Nous nous concentrons sur le modèle "smoothed-particle hydrodynamics" (SPH) qui est simple à mettre en œuvre et dont les équations peuvent être étendues pour prendre en compte de nombreux effets physiques, tels que les écoulements multiphases ou les interactions avec des objets rigides. Pour adapter la formulation SPH aux surfaces 3D, nous proposons de rassembler les voisinages des particules grâce aux géodésiques du plus court chemin, et de déplacer ces particules de manière intrinsèque sur la surface. Tout cela est facile à mettre en œuvre sur le GPU, ce qui permet de simuler des dizaines de milliers de particules sur divers maillages triangulaires à une vitesse interactive. Nous présentons les effets typiques du SPH et nos résultats sur de nombreux maillages, qui peuvent être non orientables ainsi qu'auto-intersectés, pour des fluides aux propriétés physiques variées.

En résumé, dans cette thèse nous présentons plusieurs façons de réduire les coûts des simulations de fluides par l'utilisation de la réduction de dimension. Notre première contribution utilise des modèles d'apprentissage profond, tandis que la seconde tire profit des recherches récentes en matière de modélisation de la géométrie intrinsèque.

Abstract

Computer graphics is an ever growing field, which mostly aims at producing visual content on digital media, for applications such as visual effects, video games or computer assisted design. Part of the goal of research in computer graphics is to improve the photorealism of objects and materials, or to animate characters and natural phenomena as realistically as possible. Nevertheless, despite tremendous improvements in graphics hardware performance as well as key algorithmic advancements over time, some natural phenomena remain extremely costly to simulate. For example, capturing the complex dynamics and behaviors of a fluid requires a significant amount of memory and computational resources. However, resources are available in finite amounts, thus consuming energy is expensive in terms of money, and most of all has an important impact on the environment and on human lives. For these various reasons, a lot of research has been focusing on optimizing such simulations.

Several tracks have been proposed and explored over the years to improve the performance of fluid simulations, that are typically animated by solving partial differential equations (PDE). Such equations can be extremely costly to resolve, especially in the case of fluids where the relevant PDEs are the highly non-linear Navier-Stokes equations. Therefore, it remains particularly difficult to produce real-time fluid flows at high resolutions. An important subject, that has been growing at an unprecedented pace since the beginning of the years 2000, is the use of deep neural networks in conjunction with physics-based solvers in order to reduce the computing time for solving PDEs. This data-driven approach has proven to be efficient in mimicking such solvers (or parts of them) for example to recreate very small details such as droplets. In this thesis, we first explore the use of deep learning to create a reduced space in which a solver can operate with lower costs, while still outputting high-quality solutions. Indeed, most previous works focusing on dimension reduction use a traditional bilinear down-sampling operation to reduce the degrees of freedom of a simulation, and we propose other, more relevant spaces, with no pre-defined constraint. We thus propose a model that enables the simulation of turbulent flows at a resolution four times higher than that of the given input in each dimension, with improved runtime performance compared to a high-resolution solver. This is showcased for various physical scenarios, with comparisons to traditional techniques.

In addition to using deep learning models to reduce the degrees of freedom of a simulation, we propose to create a bridge between the physics-based simulation and geometry processing communities, in order to improve the performance of fluid animation. In geometry processing, it is becoming more and more common to use intrinsic operators – that do not require a 3D embedding of the surface – as they can provide more robustness than extrinsic ones. On the other hand, one recurring challenge in fluid simulation is the handling of fluid-solid interactions. In particular, some application scenarios require solving the Navier-Stokes equations mostly on the surface, while some frameworks still simulate the whole 3D domain, which

is extremely costly. Starting from these observations, we use the important contributions on intrinsic operators for simulating fluids on 3D surfaces with reduced costs. We focus on the smoothed-particle hydrodynamics (SPH) model, which is simple to implement and whose equations can be extended to account for numerous physical effects, such as multi-phase flows or interactions with rigid bodies. To adapt the SPH formulation to 3D surfaces, we propose to gather the particles' neighborhoods thanks to shortest-path geodesics, and to displace such particles in an intrinsic manner on the surface. All of this is straightforward to implement on the GPU, enabling the simulation of tens of thousands of particles on various triangle meshes at interactive speed. We present typical SPH effects and showcase our results on numerous meshes, that can be non-orientable as well as self-intersecting, for fluids with diverse physical properties.

In summary, in this thesis we present several ways of reducing the costs of fluid simulations by the use of dimension reduction. Our first contribution uses deep learning models to do so, while the second one takes benefit from the discoveries in intrinsic geometry modeling.

Contents

Chapter 1

Introduction

Figure 1.1: The simulation of natural phenomena is often used for entertainement purposes, for example in movies. Here, a poster of the movie *Elemental* by Pixar representing the four natural elements is shown.

1.1 Context and challenges

Computer graphics is a rich and diverse field standing at the junction of Art and Science. Its applications range from computer-aided design to biomedical imaging by way of animated movies, visual effects (VFX) and video games. While it is easy to understand the usefulness of Computer Science research in a field like biomedical imaging for example, one can wonder how research can benefit artistic domains, such as animation or VFX. Nowadays, most artistic projects are rendered on digital media or created, at least drafted, via digital tools. Creators thus need to elaborate digital environments, such as a virtual scene with objects, characters, lighting, etc. Therefore, computer graphics research can provide them with relevant tools, which are usually designed in collaboration with them. Such tools can be created to answer specific needs of artists, or to enhance their creativity by widening the range of possibilities they are presented with. For that reason, part of the role of researchers in computer graphics comes down to providing artists with responsive tools that are as lightweight and intuitive as possible.

While there are probably as many artistic universes as content creators, most artistic projects are based on a certain reality common to all. Moreover, if one wants people to identify with their piece of art or story, to understand the intentions that they put in it, it needs to have at least a few common features with reality. To that end, a lot of research projects focus on reproducing real-world scenarios as accurately as possible, in environments that remain virtual. The *rendering* community works on objects' or people's appearance, whereas the *animation* community focuses on their movements or interactions. The latter is particularly important, as an unusual physical behavior can quickly take the viewer out of immersion, especially if it comes from a character that is supposed to mimic humans or animals.

Figure 1.2: Two scenes from the movie *Frozen 2* by Disney show: (left) human, animal and imaginary characters standing in a natural environment made of water, rocks and trees, and (right) a realistic simulation of a dam burst.

Realistic animation consitutes a whole branch of computer graphics, comprising character animation as well as physics-based simulation (Figure 1.1, Figure 1.2). The latter covers an extensive range of phenomena, like rigid and deformable body dynamics, fluid motion, or interactions between them. Physics-based simulations require solving partial differential equations (PDE), which is done thanks to numerical methods that usually demand a lot of computing power. Despite the exceptional advances in computing hardware since the beginning of the 21st century, simulating natural phenomena at interactive speed with limited memory

Figure 1.3: A typical 3D mesh is the Stanford bunny. Here, three different tesselations are used, from the most (left) to the least (right) refined.

remains a challenging task. The ever increasing urge for high resolution details in applications such as VFX and animation leads to computationally intensive simulations with high memory footprints. For instance, Figure 1.2 - right shows a scene from *Frozen 2* representing a dam burst, releasing huge amounts of water that are colliding with the remaining structure of the dam. This creates splashes and droplets that require a lot of resources to be animated at the desired high resolution.

In this thesis, we specifically focus on the domain of fluid simulation, which is particularly demanding in terms of computation because of the non-linearity of the Navier-Stokes equations. Indeed, modeling fluids, whether in liquid or gas form, entails solving these equations numerically for structures than can be arbitrarily complex and challenging to resolve. To tackle performance and resolution issues that arise from this complexity, one solution is to reduce the dimensionality of the problem. To do so, lots of works have focused on data-driven methods. Such methods can help circumvent the heaviest parts of the simulation, by either replacing numerical solvers or parts of them by deep neural networks (DNN). For instance, some works simulate the least costly parts of a liquid with a typical solver, while resolving the high-frequency details (such as droplets) with a DNN, in order to make performance gains on these very numerous and small details. Others on the contrary prefer to make approximations on the low frequency areas of the simulation, and concentrate their computing power on small-scale details to enable high-resolution realistic animation. It is also possible to replace an entire solver by a DNN, or to train models that learn the super-resolution of fluids in order to retrieve the details of a simulation made in a lower-resolution space.

In addition to fluids being extremely demanding in terms of computing resources, the interactions between fluids and solids can be quite complicated to model in an accurate and stable way. However, some problems involving such interactions do not require solving the equations on the whole domain. Indeed, the region of interest can be at the surface of the object, for instance when modeling viscous liquids such as paint or honey pouring or dripping down a surface, or when simulating water flowing on a window. In this case, simulating the whole 3D domain is not necessary, and restricting the computations to a surface can lead to significant performance gains. In computer graphics, the most common representation for objects is a triangle mesh (Figure 1.3) where the surface is represented as a set of 3D triangles, made of vertices interconnected by edges. A surface can be described using either an *extrinsic* or *intrinsic* approach. In the former there exists a 3D embedding of the surface, where the vertices have world-space positions and the distances between them are calculated

in the Euclidean space. The *intrinsic* description of a manifold [1] however does not require a global coordinate system, as quantities can be measured exclusively on the surface. For instance, distances between vertices in an intrinsic approach can be measured using *geodesics*, which are lines following the surface and its curves. Geodesic distances are thus equivalent to unfolding the mesh in a planar representation and applying Euclidean measurements there. One simple analogy to the intrinsic approach is the use of cardinal directions in order to find our way on Earth. These directions are defined in a 2D plane, making it easier to find our way than using a 3D curved domain. We can easily guide ourselves in this local tangential plane, in the same way as particles of a fluid can be displaced intrinsically on a surface.

1.2 Objectives and outline

In this thesis, we aim at using dimension reduction to make fluid simulation computationally lighter than with traditional numerical solvers. In Chapter 2, we present technical notions on fluid simulation, deep learning and geometry processing that we deem necessary to fully understand the content of this thesis. We then detail in Chapter 3 the main contributions introduced by these three communities over the years. In doing so, we wish to give more context and perspective on our contributions.

Thanks to the breakthrough in deep learning technologies, numerous works have been using deep learning with fluids, either for mimicking physics-based solvers that intend to solve the Navier-Stokes equations, for transforming a simulation (e.g. using super-resolution), or for simulating a specific part or feature of a fluid (e.g. droplets). Some works also focused on using deep neural networks to simulate a reduced version of a fluid, or to correct the numerical errors induced by such a reduced representation. In Chapter 4, we introduce a data-driven model that aims at outputting a high-resolution fluid simulation from a unique low-resolution frame, with minimal costs. To do so, we propose a deep neural network model that, when applied together with a low-resolution differentiable solver, outputs frames in a dimension four times larger than the input.

Our complete network, named *ATO*, is composed of three models that are trained together and optimized for a joint goal. We first transform the initial frame in an unknown and physically unconstrained latent space, using an encoder network. A step of the solver is performed on this latent state, and the next frame is adjusted to match the latent representation induced by the encoder network. This process involving the solver and the adjustment network is repeated multiple times to get a *reduced solution*. These reduced states are finally processed by a state-of-the-art super-resolution model that is trained in conjunction with the encoder and adjustment networks, in order to maximize the resemblance of the final approximated solution to a high-resolution ground truth. We demonstrate the performance of our model in complex physical scenarios, representing turbulent flows with various physical properties. We compare our model to a high-resolution ground truth, as well as to state-of-the-art works that either replace the reduced solver by a neural network, or correct the numerical errors brought on by the low-resolution at which the solver is applied.

In Chapter 5, we wish to explore the intrinsic simulation of liquids on surfaces. The smoothed-particle hydrodynamics (SPH) method has become a standard in the fluid simulation community thanks to its flexibility and simplicity. On the other hand, intrinsic modeling has

[1] A manifold of dimension 1 is a curve, and a surface is a manifold of dimension 2. This notion generalizes to dimension n.

been getting more and more appreciated in geometry processing for the robustness that it provides. Nonetheless, to the best of our knowledge no work has intended to link both yet. In this chapter, we aim at building a bridge between these two communities by using the SPH method on surfaces, thus restricting the simulation to a 2D domain in 3D space.

In SPH-based simulations, particles are equipped with scalar and vector quantities (e.g., density, pressure or velocity) averaged in local neighborhoods to compute the influence of each particle on its neighbors. The particles' motion is governed by Newton's laws, and each one of them can be treated in parallel, which makes it particularly interesting for real-time simulation. Our model conceptually assumes that each particle sees its neighborhood through a local logarithmic map on the surface and interacts with neighboring particles along shortest-path geodesics, resulting in intrinsic SPH simulations on 3D surfaces. We optimize the two canonical operators needed for this goal – neighborhood's averaging and intrinsic particle displacement – to obtain efficient parallel computations that cope with challenging inputs such as self-intersecting and non-orientable surfaces with arbitrary boundaries. We demonstrate the versatility of our approach by porting standard SPH-based effects, such as surface tension, droplets or mixing fluids with different viscosities or masses. This leads to the simulation of tens of thousands of particles at interactive speeds on high quality meshes.

We finally conclude this manuscript in Chapter 6 where we summarize our contributions on dimension reduction for fluid simulation and animation. We also develop perspectives opened by this thesis that we consider as interesting future works.

Chapter 2

Technical background

In this chapter, we introduce some key technical elements in order to make the comprehension of this thesis easier. We first focus on fluid simulation, detailing its governing equations and the data structures and simulation frameworks that are typically used. We then introduce important deep learning notions in order to correctly apprehend Chapter 4. Finally, we present the geometry processing elements that we deem essential to fully comprehend Chapter 5. The corresponding literature overview is given in the next chapter.

2.1 Fluid simulation

In this section, we first detail the fluid models and equations known as the *Navier-Stokes equations*, and then present the numerical methods that have been introduced to solve them, in particular the ones popularly used in computer graphics. In this thesis, we are particularly interested in Newtonian fluid flows, i.e. fluids whose viscosity is not affected by shear rate; water and air can be assumed of this type. In computational fluid dynamics, two different specifications are commonly employed to model Newtonian fluids. In the *Eulerian* framework, measures are made at fixed locations through which the flow is passing, whereas in the *Lagrangian* one, measures are taken on moving physical parcels. In the following, we explain how to animate fluids in both representations, which are illustrated in Figure 2.1.

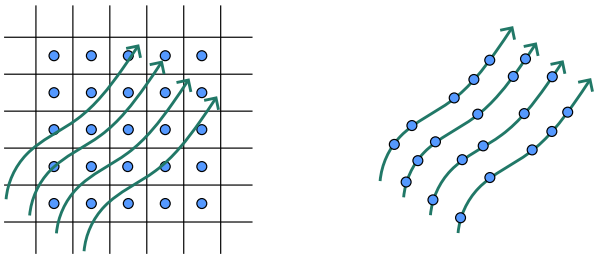

Figure 2.1: Illustration of the Eulerian (left) and Lagrangian (right) specifications for fluids. The physical quantities are measured on the blue dots.

2.1.1 Definitions

Fluid flows are described by a set of physical quantities. These can vary depending on the use-case scenario, but the following properties are most commonly used:

- The *density* ρ is defined as the ratio of the mass of fluid and its volume. It is expressed in $kg.m^{-3}$.
- The *pressure* p is the amount of force applied perpendicularly to the surface of an element per unit area. It is expressed in $kg.m^{-1}.s^{-2}$.
- The *viscosity* μ of a fluid describes its resistance to deformation. It is expressed in $kg.m^{-1}.s^{-1}$. The kinematic viscosity ν is sometimes used, with $\nu = \frac{\mu}{\rho}$.
- Finally, the *flow velocity* or *velocity field* $\mathbf{v}$ is a vector field that represents the flow direction and speed of an element of fluid at a certain position and time. It is expressed in $m.s^{-1}$.

Operators The following differential operators are commonly used for fluid simulation:

- The *gradient* of a scalar differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ gives a vector field and is written as $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$. It represents the direction and rate of fastest change of f, and is expressed as:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{2.1}$$

- The *divergence* of a vector field f, written as $\nabla \cdot f : \mathbb{R}^n \to \mathbb{R}$, represents the outgoing flux of the vector field around the point where it is evaluated. It is expressed as:

$$\nabla \cdot f = \sum_i \frac{\partial f}{\partial x_i}. \tag{2.2}$$

- The *Laplacian* of a twice differentiable scalar function f, written as $\nabla^2 f : \mathbb{R} \to \mathbb{R}^n$, is given by the divergence of the gradient of this function. It is expressed as:

$$\nabla^2 f = \sum_i \frac{\partial^2 f}{\partial x_i^2}. \tag{2.3}$$

- Finally, the *material derivative* of a vector field $f(x,t)$ depending on both time and position, in a flow that has a velocity $\mathbf{v}$, is defined as

$$\frac{Df}{Dt} \equiv \frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla f$$

  where ∇f is the *covariant derivative* of f.

2.1.2 Navier-Stokes equations

Fluid flows are modeled by partial differential equations (PDE) that describe their trajectories and behaviors, and that ensure fundamental physical principles, i.e., the conservation of mass and momentum, which are essential for their realistic simulation. Mass conservation means that the difference between the mass entering and leaving the considered volume must be null, as follows:

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot \mathbf{v}) = 0. \tag{2.4}$$

In fluid mechanics the property of *compressibility* describes how much a considered volume can see its density being modified under the action of pressure. For example, gaseous fluids are easily compressible, whereas liquids are much harder to compress. In our case, incompressible fluid flows are simulated. Their density being constant, Equation 2.4 can be simplified as:

$$\nabla \cdot \mathbf{v} = 0. \tag{2.5}$$

Newton's second law of motion states that "The change of motion of an object is proportional to the force impressed". We describe this change of motion with the *momentum* of the object, representing the product of its mass and velocity. We thus get the following equation for fluids:

$$\rho \frac{D\mathbf{v}}{Dt} = \mathbf{f} \tag{2.6}$$

with $\mathbf{f}$ representing the forces applied to the fluid. $\mathbf{f}$ comprises pressure and viscous forces, and typically includes external forces such as the gravity g. It can be written as:

$$\mathbf{f} = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{g}$$

where p represents the pressure of the fluid and μ its dynamic viscosity.

In conclusion, joining the previous equations with Equation 2.5, the movements of fluids can be described by the incompressible Navier-Stokes equations as follows:

$$\begin{cases} \rho(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v}) = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{g} \\ \nabla \cdot \mathbf{v} = 0 \end{cases} \tag{2.7}$$

2.1.3 Eulerian fluid dynamics

In the Eulerian framework, physical properties such as density, pressure or velocity are measured on grids, that can be *collocated* or *staggered* grids [49] (as illustrated in Figure 2.2). In the former, the velocity vectors are stored at the cells' centers, whereas they are stored on the cells' faces in the latter, making the computation of the divergence of a cell easier. In both cases, the scalar variables are defined at the centers of the cells. These grids are given to a numerical solver that aims at predicting the values of each physical field at the next time-step.

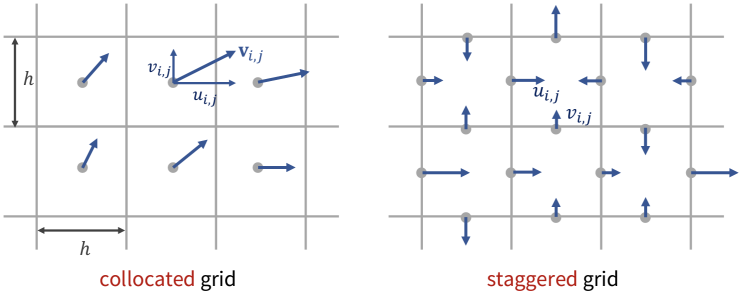

Figure 2.2: Example of a grid in an Eulerian framework, showing that the velociy values can be stored either at the cells' centers (left) or on their faces (right).

Solving method Many numerical solvers have been created to try and resolve Equation 2.7 in an Eulerian framework. Among them, we use one popular algorithm which uses Chorin's projection method with an operator splitting scheme [29]. In this method, the velocity field is first predicted by ignoring the pressure term $-\nabla p$ of Equation 2.7, which is then used to correct the predicted velocity in order to ensure divergence-free (Equation 2.5). First, the physical quantities are initialized at grid points depending on the desired initial scenario. Then, the following steps are usually executed, as described in the seminal work of J. Stam [140]:

1. Physical quantities are advected along the velocity field of the fluid,
2. External forces are applied,
3. Diffusion is applied to take viscosity effects into account,
4. The divergence-free condition is enforced.

The second step is straight-forward, but the three others have been at the heart of multiple research works. The *advection* step (1) consists in transporting the physical quantities along the velocity field of the fluid, including the velocity itself. This step is usually unstable if solved using a naive approximation of the finite difference method, leading to the use of small time-steps. Thus, the *semi-Lagrangian* method is popularly used for the advection, thanks to its unconditionally stable algorithm. Let us consider a physical quantity A at a grid point x_i at time-step t. We first imagine that we have a particle p at this location, and compute the position x where it would have been at the previous timestep $t-\Delta t$ in a Lagrangian representation. We know the values of A at grid points at this time-step, because they have been set at the previous simulation step. We can thus retrieve $A(x,t-\Delta t)$ by a simple interpolation of these values. Since we consider that p was advected from x to x_i without changing its physical properties, we get that $A(x_i,t) = A(x,t-\Delta t)$.

Then, the *diffusion* step (3) consists in solving the viscosity term $\mu\nabla^2\mathbf{v}$ of the PDE. It is possible to discretize the diffusion operator and use an explicit time-stepping scheme, as follows:

$$\mathbf{v_{t+1}} = \mathbf{v_t} + \nu\Delta t\nabla^2\mathbf{v_t},$$

but it can suffer from numerical instabilities if the viscosity coefficient or time-step size is too large or if the density is too small. This can be addressed by using an implicit scheme, which

leads to a sparse linear system that can be solved using an iterative linear system solver:

$$(I - \nu \Delta t \nabla^2)\mathbf{v_{t+1}} = \mathbf{v_t}.$$

Finally, the most expensive step in fluid solvers is the *projection* step (4), which consists in ensuring that the fluid is divergence-free, as the previous steps may have created undesirable behaviors. Indeed, the velocity obtained after the diffusion step is usually not divergence-free, and needs to be projected into a divergence-free field, by using a pressure field that is retrieved from the Poisson equation thanks to an iterative linear system solver:

$$\begin{cases} \mathbf{v_{t+1}} = \mathbf{v_t^*} - \frac{\Delta t}{\rho} \nabla p^{t+1} \\ \nabla^2 p^{t+1} = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{v_t^*} \end{cases}$$

where $\mathbf{v}^*$ represents the intermediate velocity field that was predicted without accounting for the pressure forces.

Boundary conditions The steps that are described above are performed in a simulation domain; thus, it is essential to define boundary conditions. Mainly two types are considered: Dirichlet and Neumann boundary conditions. The choice of the boundary conditions depends on the use case. For instance, we might want a flow to evolve in a box that is opened at one side but closed on the others. Accordingly, we may apply a Dirichlet boundary condition for the velocity, setting the velocity values at the closed sides to zero such that no flux through the boundary is allowed. On the other hand, we may apply a Neumann boundary condition for the pressure in the Poisson equation, setting the pressure gradient at the boundary to zero such that no pressure difference is allowed. Additionally, a periodic boundary condition can be considered such that the one side of the domain is connected to the other side.

2.1.4 Lagrangian fluid dynamics

In the Lagrangian specification for fluids, physical quantities are measured on individual parcels. In this framework, the Navier-Stokes equations (Equation 2.7) can be considerably simplified. Indeed, since we measure physical properties on the fluid parcels that we follow, the material derivative of the velocity is equivalent to the simple time derivative:

$$\frac{D\mathbf{v}}{Dt} \equiv \frac{d\mathbf{v}}{dt}.$$

Therefore, if we regroup the pressure, viscosity and external forces in a single term

$$\mathbf{f} := \mathbf{f}^{pressure} + \mathbf{f}^{viscosity} + \mathbf{f}^{external}, \tag{2.8}$$

Equation 2.6 gives for the update of the velocity field:

$$\frac{d\mathbf{v}}{dt} = \frac{\mathbf{f}}{\rho}.$$

In this thesis, we focus on a specific method to simulate Lagrangian fluids, namely smoothed-particle hydrodynamics (SPH) [98]. This technique relies on a set of equations

that enable the interpolation of physical quantities anywhere in space. In the following, we describe the most basic SPH method for 2D domains. We note that this SPH formulation does not allow for the simulation of incompressible fluids and can lead to a poor approximation and unstable simulations. Therefore we refer the reader to Chapter 3, Section 3.1 for a presentation of more recent and better approximations.

Let A denote a scalar quantity that we wish to measure, such as density. The value of A is stored for each particle, and it can be evaluated at any given position x by taking a weighted average of its value at neighboring particles in a radius h, with a weighting kernel W_h. These particles contribute depending on their position x_j, their mass m_j and their density ρ_j, as follows:

$$A(x) := \sum_j \frac{m_j}{\rho_j} A(x_j) W_h(x - x_j). \tag{2.9}$$

If a quantity A has to be differentiated with respect to space, as it is the case for example to compute some forces, only W_h in Equation 2.9 is differentiated, as follows:

$$\nabla A(x) := \sum_j \frac{m_j}{\rho_j} A(x_j) \nabla W_h(x - x_j)$$

or

$$\nabla^2 A(x) := \sum_j \frac{m_j}{\rho_j} A(x_j) \nabla^2 W_h(x - x_j).$$

With these equations, we can compute the different components of the force term described in Equation 2.8, making sure that the particles have a symmetrical influence on each other as described by Netwon's third law. Firstly, if we used this equation in a straightforward manner to define the pressure force $-\nabla p$ applied to a particle i by its neighborhood, we would get

$$\mathbf{f}_i^{pressure} := \sum_j m_j \frac{p_j}{\rho_j} \nabla W_h(x_i - x_j)$$

which would not give the desired symmetrical behavior. Thus, it is made symmetrical by using the average of the particles' pressure, and it can be expressed as:

$$\mathbf{f}_i^{pressure} := \sum_j m_j \frac{p_j + p_i}{2\rho_j} \nabla W_h(x_i - x_j). \tag{2.10}$$

Similarly, the viscosity force $\mu \nabla^2 \mathbf{v}$ applied to a particle i is expressed as:

$$\mathbf{f}_i^{viscosity} := \mu \sum_j m_j \frac{\mathbf{v_j} - \mathbf{v_i}}{\rho_j} \nabla^2 W_h(x_i - x_j). \tag{2.11}$$

Finally, the surface tension can be estimated as an additional force. It models how liquids occupy the minimal area possible when interacting with air for example. First, the surface normal is estimated as:

$$\mathbf{n} := \sum_j \frac{m_j}{\rho_j} \nabla W_h(x_i - x_j)$$

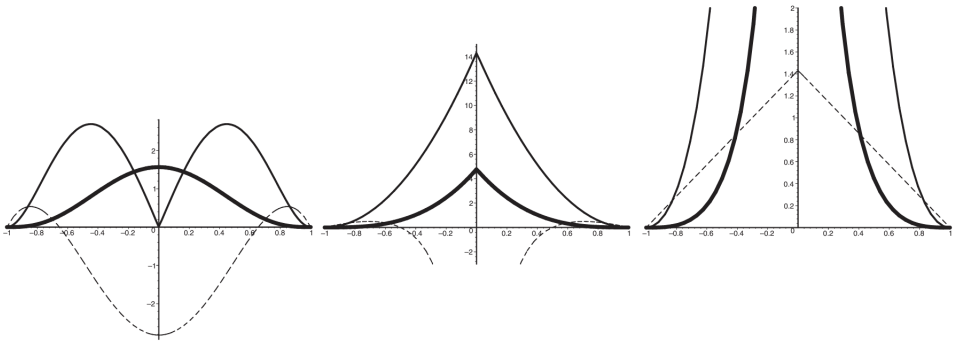

Figure 2.3: The three kernels $W^{default}$, $W^{pressure}$ and $W^{viscosity}$ (thick lines) that are typically used for SPH simulations are shown along their gradients (thin lines) and Laplacians (dashed lines).

and its divergence represents the curvature of the surface. The surface tension is thus described by the following equation in the SPH framework:

$$\mathbf{f}_i^{surface} := -\sigma \frac{\mathbf{n}}{|\mathbf{n}|} \sum_j \frac{m_j}{\rho_j} \nabla^2 W_h(x_i - x_j) \tag{2.12}$$

with σ a tension parameter.

Smoothing kernels Specific smoothing kernel functions can be used depending on the desired quality, stability or performance. In this manuscript, we use the kernels from [101] (shown in Figure 2.3) adapted for 2D as follows:

- The default kernel is:

$$W_h^{default}(r) := \frac{315}{64\pi h^8} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & otherwise \end{cases}$$

- For pressure forces we use:

$$\nabla W_h^{pressure}(r) := \frac{-45}{\pi h^5} \frac{\mathbf{x}}{\|\mathbf{x}\|} \begin{cases} (h-r)^2 & 0 < r \le h \\ 0 & otherwise \end{cases}$$

  where $\mathbf{x}$ is the vector pointing to the evaluation point from the center of the kernel, thus $\|\mathbf{x}\| = r$.

- For viscosity forces we use:

$$\nabla^2 W_h^{viscosity}(r) := \frac{45}{\pi h^5} \begin{cases} h - r & 0 \le r \le h \\ 0 & otherwise \end{cases}$$

ALGORITHM 1: A typical step of the smoothed-particle hydrodynamics algorithm.

for *each particle i* **do**
  compute neighborhood
end
for *each particle i* **do**
  $\rho_i = \sum_j m_j W_h(x_i - x_j)$
  $p_i = k(\rho_i - \rho_0)$ `with k the stiffness parameter`
end
for *each particle i* **do**
  Compute $\mathbf{f}_i = \mathbf{f}_i^{pressure} + \mathbf{f}_i^{viscosity} + \mathbf{f}_i^{surface} + \mathbf{f}_i^{external}$
  $\mathbf{v}_{i,t+1} = \mathbf{v}_{i,t} + \frac{\Delta t}{\rho_i} \times \mathbf{f}_i$
  $\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} + \Delta t \times \mathbf{v}_{i,t}$
end

Solving method In Algorithm 1 we describe one typical step of the most basic SPH method, as proposed by Müller and colleagues in [101]. First, at each time-step the neighborhood of the particles must be set. The neighborhood of a particle i at position x_i is defined as all the particles lying inside a circle of radius h (the kernel smoothing radius) and center x_i. To find the particles' neighbors, hash tables are typically used to prevent going over n^2 particles at each step (with n the total number of particles). Particles are usually sorted spatially, and only the ones inside a restricted bounding volume are checked. In Chapter 5, we detail the specific neighborhood structures that we employ for our intrinsic SPH implementation.

After the neighborhoods of the particles are set, we can compute the density and pressure values of each particle thanks to Equation 2.9. We can then compute the forces acting on them thanks to Equation 2.10, Equation 2.11, and Equation 2.12. Finally, the position and velocity of each particle can be updated using a numerical integration scheme such as the Euler method. The time step used for the numerical integration must obey to the Courant–Friedrichs–Lewy condition as follows, to ensure convergence:

$$\Delta t \leq C \times \frac{d}{\|\mathbf{v_{max}}\|}$$

where C is called the CFL number (often set to 0.4 for SPH), d is the diameter of a particle and $\mathbf{v_{max}}$ the maximum velocity of the flow.

Boundary conditions Finally, boundaries in SPH are in general handled using several layers of frozen particles placed at the borders that are taken into account in the forces calculation, but stay static during the simulation. This can cause some fluid particles to go through the boundary, which has been tackled for example by Ihmsen and colleagues in [59] by using a prediction-correction scheme.

2.2 Deep learning

In Chapter 4, we present a hybrid method that interleaves deep learning with a physics solver. In this section, we thus wish to introduce deep neural networks and how they work. We first present the seminal multilayer perceptron model, and then introduce the widely used convolutional neural networks. A more thorough review of the literature in deep learning methods is presented in Chapter 3.

2.2.1 Multilayer perceptron

Deep learning is a branch of machine learning, itself being a subdomain of artificial intelligence. The goal of machine learning algorithms is to make decisons in an autonomous fashion, aiming at the achievement of a specific task, which can be for example classification, segmentation or content generation. A subclass of these algorithms are *neural networks*, which are composed of layers of *neurons* that are stacked together. Each neuron can be activated under certain conditions, which will influence the output of the algorithm. The particularity of such networks is that they are capable of distinguishing data that is not linearly separable. These neural networks operate by first learning features from very large amounts of data in a training phase, in order to then extrapolate from other inputs at inference time.

The first neural network model was introduced in 1958 by Rosenblatt [121], and was composed of three layers of neurons: one input, one output and one intermediate hidden layer. Later, other works proposed to use more than one hidden layer, giving birth to the multilayer perceptron (MLP), of which an example architecture is shown in Figure 2.4. A neural network is said to be a *deep neural network* (DNN) if composed of more than three layers.

Let us denote as $x \in \mathbb{R}^n$ the input signal received by a neuron, we can write its output as:

$$f_{\theta,b}(x) = \sigma(\theta^T x + b) \tag{2.13}$$

where $\theta \in \mathbb{R}^n$ is the weight vector of the neuron, $b \in \mathbb{R}$ is its bias and σ is a non-linear function called the *activation* function. The combination of neurons outputs a value that is

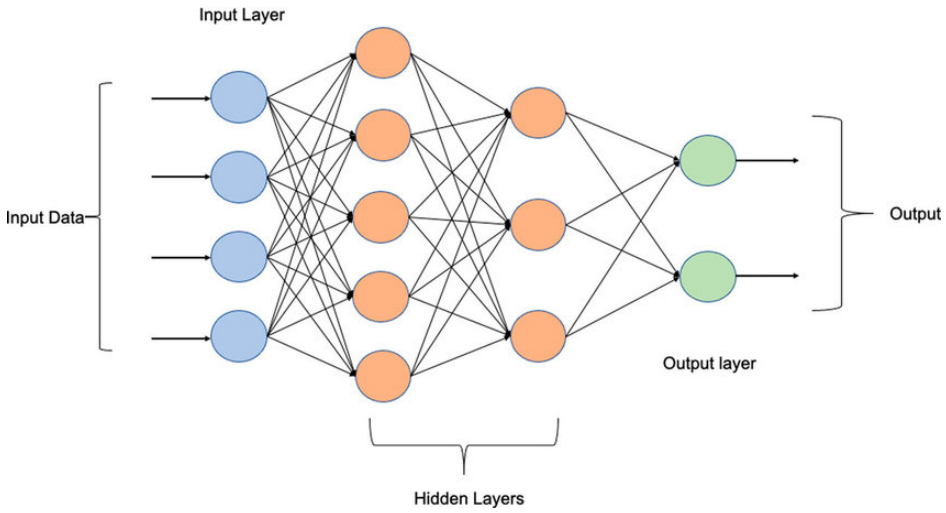

Figure 2.4: Example architecture of a multilayer perceptron with two hidden layers from [2]. The circles represent the neurons.

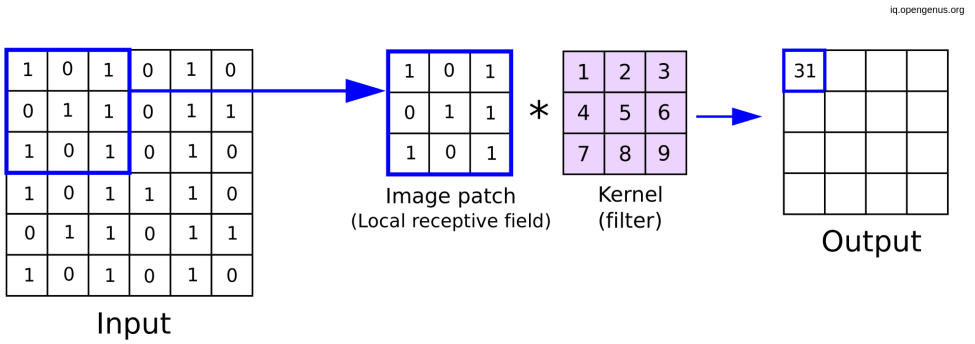

Figure 2.5: Example of convolutional layer. The kernel is applied to a small patch of the input image, resulting in an output that has the same size as the input before padding was applied.

compared to a reference, thanks to a *loss* function. The parameters (weight and bias) of the neurons are optimized during training, with the goal to minimize the average error between the model prediction and the ground truth, over the training dataset. To run this optimization, a backpropagation is performed over the network's neurons. To that end, the stochastic gradient descent (SGD), introduced in 1951 [118, 70] is generally used. This algorithm runs a gradient descent iteratively to update the weights of the neurons, starting from a random initialization of the parameters. This method, by introducing noise in the gradients' estimation, prevents getting stuck in local minima. Later, the now extensively used ADAM optimizer [75] has been proposed, introducing momentum as well as a decaying learning rate to the SGD optimization method, thus accelerating its convergence rate.

Finally, the activation functions play an important role in the network optimization. The rectified linear unit (ReLU), defined as $\sigma(x) = \max(0,x)$, has been used extensively. However, its null derivative for negative inputs led to "dying neurons" – i.e. neurons that are never activated for a wide range of input values – during the backpropagation process. To that end, the Leaky ReLU function has been introduced, defined as $\sigma(x) = \max(\varepsilon x, x)$, with $\varepsilon \in]0,1[$. It presents the advantage of having a strictly positive derivative when its input is strictly negative.

2.2.2 Convolutional neural networks

In image processing, other types of DNNs are generally used, namely convolutional neural networks (CNN). Indeed, although MLPs proved their efficiency in many fields, they are not very well suited for large data such as images, for which they are unnecessarily heavy. For that reason, Le Cun and colleagues introduced CNNs in 1989 [83]. These models rely on convolutional kernels (of which an example is shown on Figure 2.5), which can be seen as filters that are applied all over the input image, leading to the creation of feature maps. These operations being fully differentiable, the typical training and optimization process described above can be implemented. Since convolutions have the particularity of being translation-invariant, these models are frequently used for their capabilities in detecting patterns and redundancies in an image.

In order to account for the boundary of the input image, a padding is typically applied before each convolution. The padded values can be zero, or they can be a repetition of the boundary values. In some cases where periodicity or self-similarity need to be enforced, a periodic padding can be implemented. Additionally from convolutions, pooling layers are

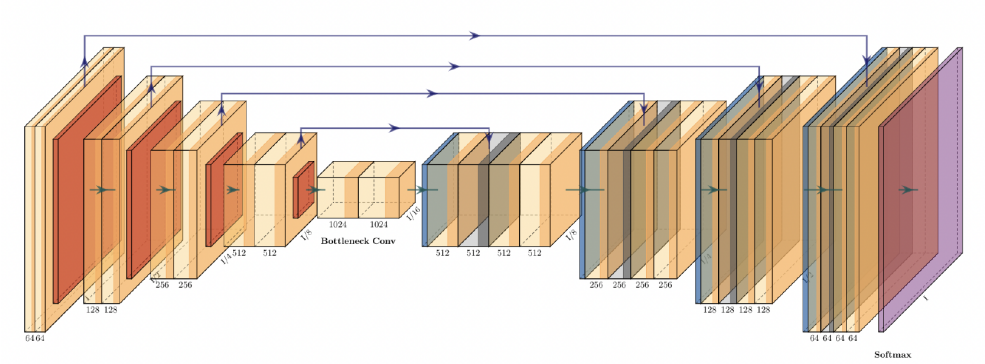

Figure 2.6: Architecture of the U-Net model proposed by Ronneberger and colleagues in [120]. This model is composed of an encoder, a decoder, and skip-connections concatenating feature maps from the encoder to maps from the decoder. This figure was taken from [1].

commonly applied in CNN architectures in order to reduce the dimension of intermediate feature maps. The most frequently used pooling layers are *max-pooling* and *average pooling* layers, respectively keeping the maximum or average value in a given window. These are particularly used when the network is expected to output a single value, for example for classification algorithms.

In Chapter 4, the deep learning model that we propose is entirely composed of CNNs, partly in the form of *encoder* and *decoder* networks. Such models, which can be seen as a series of convolutional and pooling layers, are usually linked by a *latent space*. A latent space, or latent feature space, is an embedding where elements are placed depending on their resemblances. It is usually of lower dimensionality than the original input feature space. A first typical example of encoder-decoder architecture is the *autoencoder*. An autoencoder comprises an encoder that transforms the input data into a lower dimensional latent space, and a decoder that aims at recreating the input from this latent representation. Such models are widely used for text processing, as well as for image compression, denoising or generation applications. Another famous encoder-decoder architecture is the U-Net model, first introduced for image segmentation in biomedical imaging. In this model (of which an architecture is shown on Figure 2.6), the dimension of the input is first reduced thanks to the encoder network, and the encoded representation is then transformed back to the original input dimension, with the help of intermediate *skip-connections*. These skip-connections concatenate intermediate feature maps from the encoding phase to maps from the decoding phase, enabling the recovery of compressed information.

Finally, encoder-decoder CNNs have been used extensively in the context of image synthesis. Three architectures particularly stand out in this field, namely *generative adversarial networks* (GAN), *variational autoencoders* (VAE) and *diffusion probabilistic models* Firstly, GANs (first row of Figure 2.7) are composed of two neural networks: a generator, that learns to synthesise high-quality outputs from a latent vector sampled from a normal distribution, and a discriminator, that learns to distinguish between real and synthetic data. The loss of a GAN is said to be adversarial because both elements are competing against each other. Such models can be particularly difficult to train, because their goal is not to simply minimize one loss. However, in cases where this adversarial training is a success, GAN architectures allow for the generation of diverse, high-quality images. Secondly, variational autoencoders

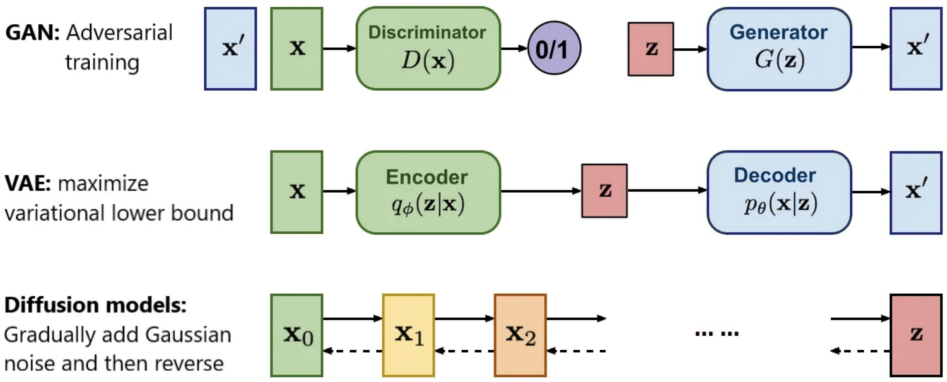

Figure 2.7: Coarse architectures of the GAN, VAE and diffusion models. This figure was taken from [36].

(second row of Figure 2.7) have a typical autoencoder architecture, with the difference that their latent space can be seen as a probabilistic distribution. Therefore, instead of mapping the input data to a single point, the encoder of a VAE outputs the normal distribution of the latent variable. This method is easy to train and leads to very diverse generation samples, but often gives blurry outputs. Lastly, diffusion models (third row of Figure 2.7) are operating in two phases. First, a forward pass is performed during which Gaussian noise is gradually added to the input data, until it becomes white noise. Then, a neural network learns to remove this noise in order to recover the original input, with the same number of steps as the forward pass. Diffusion models are easy to train and allow for the synthesis of diverse and high-quality samples. However, they are much slower to train than GANs or VAEs.

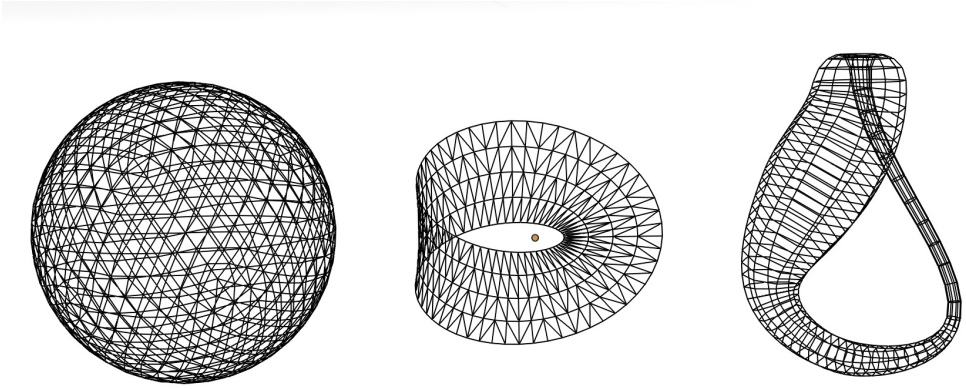

Figure 2.8: Three manifold meshes are shown: (left) an orientable mesh of a sphere, (middle) a non-orientable Moebius strip and (right) a non-orientable and self-intersecting Klein bottle.

2.3 Intrinsic geometry processing

In this section we introduce the technical tools that are necessary to correctly approach Chapter 5, in which we apply geometry processing methods to fluid simulation. We first describe how a mesh is typically represented, detailing the mathematical formalism that we use in Chapter 5, and then present the algorithm that we base our work on to compute geodesic distances.

2.3.1 Mesh representations

In computer graphics, surfaces are often represented by triangle meshes, which are composed of triangles interconnected by their edges and/or vertices. If the neighborhood of every point of a mesh is homeomorphic to a disk (or half a disk for boundary points), it is said to be *2-manifold*. In this case, every edge is shared by at most two faces. To define if a mesh is also *orientable*, we must:

1. define an order to enumerate the triangles' vertices,

2. check that for each pair of triangles sharing an edge (i, j), i and j appear in opposite order.

Such orientable meshes (see Figure 2.8 - left) represent real-life objects since they have a clearly defined interior and exterior. However, in geometry processing, non-orientable meshes also present interesting mathematical properties. In our case, interesting non-orientable geometry would for example be the famous Moebius strip and Klein bottle, which is also self-intersecting, showed in Figure 2.8 - middle and right.

In this manuscript we focus on meshes whose triangles are connected through the edges only, and not through the vertices. We introduce the mathematical notations and geometrical formalism that enable us to manipulate points and vectors on such surfaces, as needed for our contribution on intrinsic fluid simulation.

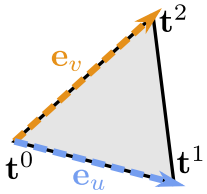

Figure 2.9: Triangle $t = (\mathbf{t}^0, \mathbf{t}^1, \mathbf{t}^2)$, with $\mathbf{e}_u := \mathbf{t}^1 - \mathbf{t}^0$ and $\mathbf{e}_v := \mathbf{t}^2 - \mathbf{t}^0$

Canonical barycentric coordinates for positions and vectors We identify a 3D **point p** on a surface $\mathscr{S}$ using the index t of the triangle on which $\mathbf{p}$ lies, along with its *barycentric coordinates* (u,v) inside $t = (\mathbf{t}^0, \mathbf{t}^1, \mathbf{t}^2)$ (see Figure 2.9):

$$\mathbf{p} = \mathbf{t}^0 + u\underbrace{(\mathbf{t}^1 - \mathbf{t}^0)}_{=:\mathbf{e}_u} + v\underbrace{(\mathbf{t}^2 - \mathbf{t}^0)}_{=:\mathbf{e}_v} = \underbrace{(1-u-v)}_{=:\alpha_0}\mathbf{t}^0 + \underbrace{u}_{=:\alpha_1}\mathbf{t}^1 + \underbrace{v}_{=:\alpha_2}\mathbf{t}^2.$$

We define the 3D normal of an input triangle t as

$$\mathbf{n}_t = \frac{\mathbf{N}_t}{\|\mathbf{N}_t\|}, \text{ with } \mathbf{N}_t := \mathbf{e}_u \times \mathbf{e}_v.$$

Given a 3D **vector** $\boldsymbol{\delta}$, we extract its tangential projection $\boldsymbol{\delta}_t$ on t by removing its normal component:

$$\boldsymbol{\delta}_t := (I - \mathbf{n}_t \mathbf{n}_t^T)\boldsymbol{\delta}.$$

Like for 3D positions, we associate tangent vectors with (δ_u, δ_v) coordinates inside t as

$$\begin{pmatrix} \delta_u \\ \delta_v \end{pmatrix} = (E_t^T E_t)^{-1} E_t^T \boldsymbol{\delta}_t =: P_t \boldsymbol{\delta}_t,$$
$$\text{with } E_t := (\mathbf{e}_u | \mathbf{e}_v) \in \mathbb{R}^{3\times 2}$$

where $P_t \in \mathbb{R}^{2\times 3}$ allows expressing any 3D tangent vector in t as a combination of t's edges: $\boldsymbol{\delta}_t = \delta_u \mathbf{e}_u + \delta_v \mathbf{e}_v$.

One can compute the (u,v) coordinates of a point $\mathbf{p} \in t$ as

$$\begin{pmatrix} u \\ v \end{pmatrix} =: c_t(\mathbf{p}) = P_t(\mathbf{p} - \mathbf{t}^0).$$

Using this simple formalism, translating a point $\mathbf{p}$ inside t can equivalently be performed in 3D or in (u,v) coordinates, as

$$c_t(\mathbf{p} + \lambda \boldsymbol{\delta}_t) = c_t(\mathbf{p}) + \lambda P_t \boldsymbol{\delta}_t$$

as long as $\mathbf{p} + \lambda \boldsymbol{\delta}_t$ remains inside t – or equivalently:

$$\begin{cases} 0 \le u' \le 1 \\ 0 \le v' \le 1 \\ 0 \le 1 - u' - v' \le 1 \end{cases}, \text{ with } (u', v') = (u, v) + \lambda(\delta_u, \delta_v).$$

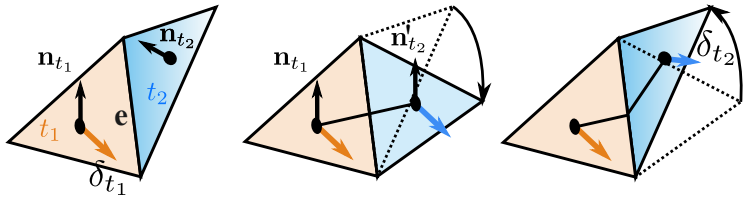

Figure 2.10: Transport of a vector δ_{t_1} from a triangle t_1 to another triangle t_2 thanks to $T_{t_1t_2}$.

Trivial connections between adjacent triangles Given two adjacent triangles t_1 and t_2 sharing an edge e (see Figure 2.10), we define $f(t_1;t_2) = f(t_2;t_1) \in \{-1;+1\}$ indicating whether they are oriented consistently or not (e.g., $f((a,b,c);(a,b,d)) = -1$, $f((a,b,c);(b,a,d)) = +1$). Following previous work on trivial connections [116, 31], we define a rotation matrix $\mathscr{T}_{t_1t_2}$ mapping tangent vectors in t_1 to tangent vectors in t_2.

$$\begin{aligned}
\mathscr{T}_{t_1t_2} &:= B_2 B_1^T, \text{ with} \\
B_1 &:= (\mathbf{e} | \mathbf{n}_{t_1} | \mathbf{e} \times \mathbf{n}_{t_1}) \in \mathbb{R}^{3\times 3} \\
B_2 &:= \left(\mathbf{e} | \mathbf{n}'_{t_2} | \mathbf{e} \times \mathbf{n}'_{t_2}\right) \in \mathbb{R}^{3\times 3} \\
\mathbf{n}'_{t_2} &:= f(t_1;t_2)\mathbf{n}_{t_2}.
\end{aligned}$$

This transport operation can be seen as (see Figure 2.10):

1. Rotating t_2 along the shared edge $\mathbf{e}$ to align $\mathbf{n}'_{t_2}$ onto $\mathbf{n}_{t_1}$;
2. Transporting $\boldsymbol{\delta}_{t_1}$ into the aligned t_2;
3. Rotating t_2 back to its original position, with the transported tangent vector.

Note that $\mathscr{T}_{t_2t_1}\mathscr{T}_{t_1t_2} = I$ as transporting vectors from t_1 to t_2, and immediately back to t_1, preserves them.

Using those notations, one can transport a vector from t_1 to t_2 in (δ_u, δ_v) coordinates using the following 2D map:

$$T_{t_1t_2} := P_{t_2}\mathscr{T}_{t_1t_2}E_{t_1} \in \mathbb{R}^{2\times 2}$$

This map remains invertible as long as the two triangles are not *degenerate* (i.e., $\text{Rank}(E_{t_1}) = \text{Rank}(E_{t_2}) = 2$), but it is no longer a rotation matrix in general.

2.3.2 Geodesic distances

A *geodesic* is a curve representing the shortest path between two points on a surface, and it is often described as the generalization of a straight line on a manifold. Geodesics are used in diverse fields – like physics, geography or computer graphics – for example to compute the shortest path between the vertices of a triangle mesh. In this thesis, we choose the algorithm proposed by Mitchell, Mount, and Papadimitriou (MMP) in [95] to compute geodesic distances (i.e. finding the distance between points by following geodesics) on triangle meshes. An example of geodesic paths and corresponding isolines from the paper of Surazhsky and colleagues [143] using this algorithm is shown in Figure 2.11.

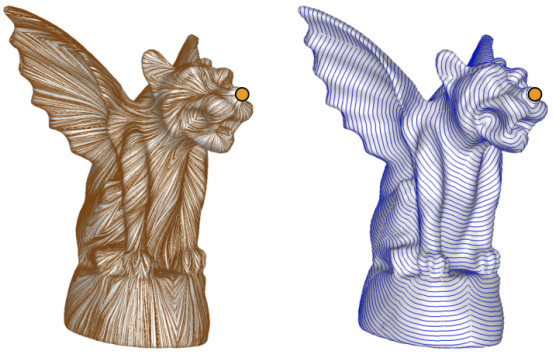

Figure 2.11: Geodesic paths from a source point (orange) and corresponding isolines from [143].

Geodesic windows In order to find the geodesic distance between a *source point* **p** and any other point of the mesh, the MMP algorithm starts by propagating *geodesic windows* from **p** all over the mesh. To create these windows, we start from the triangle containing **p** and unfold the geometry so that it becomes planar. In such a *planar unfolding*, the geodesic and euclidean distances become equivalent. The geodesic window associated to a point **p** and an edge **e** is described by a sextuplet $(b_0, b_1, d_0, d_1, \tau, \sigma)$ where b_0 and b_1 are the window's start and end positions along **e**, d_0 and d_1 are the geodesic distances from **p** to b_0 and b_1 and τ is the direction perpendicular to **e** pointing away from **p**. σ is the geodesic distance between **p** and a potential initial source, which we will clarify in the following. The geodesic window associated to **p** and **e** represents the portion of **e** for which the accumulation of geodesics giving the smallest total distance from **p** comes from the same planar unfolding (Figure 2.12).

The window propagation goes as follows:

- Given a window on an edge **e** and t the neighbor triangle towards which τ is pointing, one can compute the propagation of this window on the two other edges of t. As shown in Figure 2.13 this can be done using line-segment intersection operators. If the window touches both facing edges then it generates two windows, one on each edge, as the planar unfolding is different for the two neighboring triangles. This creates a possibility to have as many as n^2 windows for a single source point (with n the total number of triangles). However, since windows not holding the smallest accumulated distance on an edge are not propagated, the real number of handled windows is much smaller (see [143]).

- This propagation mechanism admits a special case around saddle points **q**. Indeed, if we sum the angles between the edges intersecting at **q**, the total can be more than 2π. This creates a shadowing effect from the source, as described in Figure 2.13 - (d), since no planar unfolding containing the source and **q** exist. Handling this case requires to restart a propagation process from **q**, acting as an auxiliary source, inside the shadow. In practice, this means that if a window has a saddle point **q** as an extremity, we propagate extra windows in its shadow with **q** as a new source and σ increased by the distance from the previous source to **q**.

- When inserting a new window w on an edge, we compare it against accumulated windows $\{\tilde{w}\}$ and keep only parts of w and $\{\tilde{w}\}$ that have the smallest distance to the source.
- Implementing this method with floating point arithmetics leads to well-documented numerical precision issues. We refer to [89] for a robust implementation.

The MMP algorithm starts from windows covering the three edges of the source triangle and propagates in a breadth-first manner until convergence. In our implementation, we adopt a conservative saddle point detection strategy, as false positives only degrade runtime or memory performance, whereas false negatives result in wrong outputs. Once we have the geodesic windows from a source point, we can compute the geodesic distance from the source to any point covered by these windows, thanks to the given planar unfolding and the knowledge of d_0 and d_1.

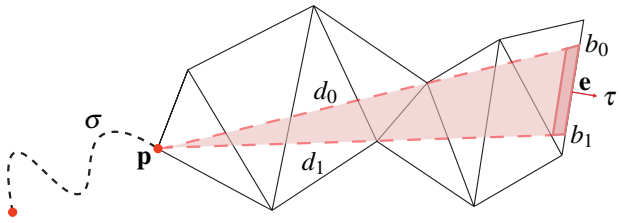

Figure 2.12: A geodesic window $[b_0, b_1]$ is associated with a common planar unfolding: the straight dotted lines connecting (b_0, b_1) to the vertex $\mathbf{p}$ delimit an area (red) containing no other vertex. A potential initial source can be present at a geodesic distance σ of $\mathbf{p}$.

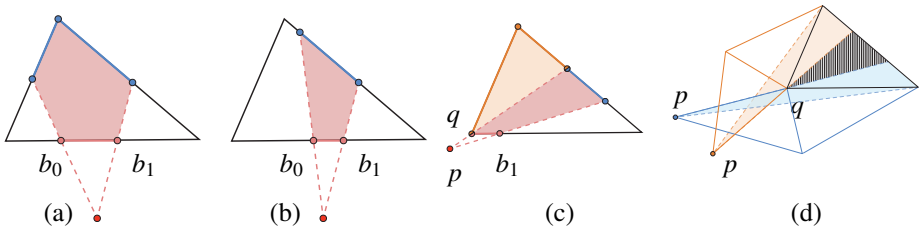

Figure 2.13: Vanilla window propagation cases (a) and (b) create one or two windows depending on whether the opposing vertex is inside or outside the window. When a window touches a saddle point q (c) two extra windows (orange) are created with the saddle point as source and $\sigma' = d_0$ or $\sigma' = d_1$. (d) In this case, planar unfoldings (blue and orange triangles) around the saddle $\mathbf{q}$ lead to different realigned positions of the primary source $\mathbf{p}$, generating windows that do not reach each other. The hashed area is the shadow of the saddle point.

We introduced some basic notions on fluid simulation, deep learning models and geometry processing. In this thesis, we will first use deep learning models in conjunction with a fluid solver in order to explore physical latent spaces for turbulent flow restoration. Then, we will compute geodesic paths and distances on triangle meshes, to apply smoothed particle hydrodynamics in order to simulate fluid flows on 3D surfaces.

Chapter 3

Related work

In this thesis, we focus on the numerical simulation of fluids, which has been at the heart of numerous research projects in the fields of computational fluid dynamics (CFD) and computer graphics since the middle of the 20th century. A lot of different techniques have been proposed since then, and we propose to present the most important and most used ones in this chapter. Since our first contribution is a hybrid method using both fluid simulation and deep learning, and the second one is using geometry processing, we introduce the necessary state-of-the-art methods in these fields too.

3.1 Fluid simulation and animation

Fluid animation is a vast field, comprising many important methods that aim at reproducing the behavior of fluids as accurately and with the least computations as possible. These methods can be divided in three categories: Eulerian, Lagrangian and hybrid methods. In the following, we present the main techniques proposed in these three categories. The technical details on how to solve the fluid equations in each representation have been given in Chapter 2.

3.1.1 Eulerian specification

In the Eulerian specification, the physical quantities that describe the fluid flow are stored on a grid. Harlow and Welch introduced the *staggered grid* as a data structure used for their marker-and-cell (MAC) method [49], which proves to be very useful when solving for incompressibility as it makes central differences more accurate. However, such grids have the drawback of being static and not well suited for a flow that would not be equally distributed over space, thus showing poor memory usage and performance. Sparse blocked grids have been proposed to tackle this issue [23, 84, 104].

The solving algorithm associated to Eulerian frameworks comprises several steps (detailed in Chapter 2) that evolved as new research propositions arose. Firstly, to circumvent the unstability issues created by explicit advection schemes, Stam introduced the *semi-Lagrangian* method to computer graphics in 1999 [140]. Since then, many works have been published to improve the accuracy of this advection step. Kim and colleagues first proposed to apply the "back and forth error compensation and correction" method (BFECC) to fluid advection in [71]. Selle and colleagues then presented an unconditionally stable Mac Cormack advection method in [127]. Their method reduces the cost of BFECC while enabling second order accuracy and

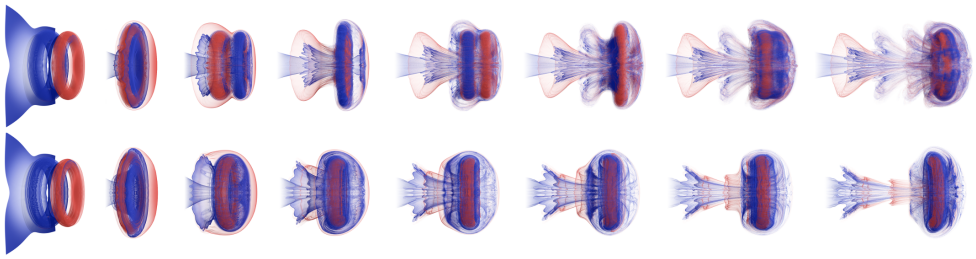

Figure 3.1: Results from "An Advection-Reflection Solver for Detail-Preserving Fluid Simulation" [164]. The top row was made using their advection-reflection step, whereas the bottom row used a typical Mac Cormack advection-projection step [127]. The two vortices stay separated despite moving through each other in the top simulation, contrary to the bottom one.

unconditional stability. Finally, Zehnder and colleagues introduced an advection-reflection method in [164] (Figure 3.1). They proposed to replace the usual projection step, that tends to uncorrectly dissipate energy at the end of each simulation step, by a reflection step applied at mid-time, that preserves energy.

Secondly, the main challenge with solving the Navier-Stokes equations numerically is to perform the incompressibility step in a reasonable amount of time. This is often equivalent to solving a Poisson equation, that leads to a sparse, symmetric and positive-definite linear system, which many optimization methods – such as the Jacobi or Gauss-Seidel methods – can solve. However, these are usually not well-suited for real-time applications or tend to show poor convergence, and are generally computationally intensive. Mac Adams and colleagues proposed to use multigrids on irregular voxelized domains with a mix of Dirichlet and Neumann boundary conditions in [94], in order to accelerate this projection step. On the other hand, Setaluri and colleagues leveraged in [128] the acceleration structures of graphics hardware to propose a novel data structure for cartesian grids along with adaptive discretizations and solvers.

Eulerian schemes have several drawbacks, in particular when it comes to rendering the fluid after its simulation. Such frameworks also suffer from sampling and conservation artifacts, that can be addressed with adaptive sampling techniques [90] but remain complex and costly to implement. Eulerian frameworks are also not well suited for dynamic boundary cases because of their lack of adaptivity. To tackle these issues, some prefer the Lagrangian specification, that relies on particles to represent the fluid flow.

3.1.2 Lagrangian specification

In Lagrangian representations of a fluid flow, the physical quantities that are measured and that are advected by the flow velocity are stored on individual parcels. To simulate their movement and interactions, the smoothed-particle hydrodynamics (SPH) method was proposed. It was first introduced to the field of astrophysics [98], and then applied to the simulation of highly deformable bodies by Desbrun and Cani [35]. It was finally extended to the animation of fluids with free surfaces by Müller and colleagues [101].

Since then, various contributions have been made to enhance the SPH method. One challenge when simulating a fluid with SPH is to ensure its incompressibility. Many global pressure solvers – such as PCISPH [137], IISPH [60] and DFSPH [14] (Figure 3.2) – have been proposed to improve the results from the original work of Müller, which struggled to

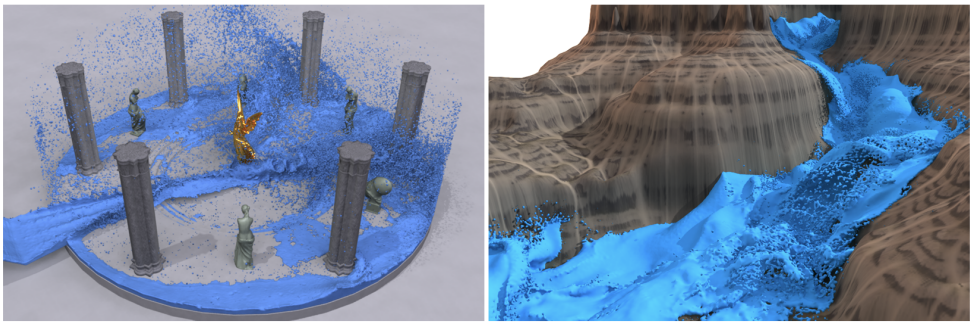

Figure 3.2: State-of-the-art incompressible SPH simulations from "Divergence-Free SPH for Incompressible and Viscous Fluids" (DFSPH) by Bender and Koschier [14].

reproduce incompressible flows. All of these solvers rely on a prediction-correction scheme, where a temporary velocity field is predicted without accounting for the pressure forces. Then, this velocity is corrected by enforcing its divergence-free and a constant density.

Secondly, many works have introduced different boundary handling methods. The most popular one consists in creating additional particles that model the boundaries of the fluid domain [59, 12, 11]. Some works employ a single layer of boundary particles while others use multiple layers, and the sampling of the boundary can be uniform or non-uniform [3]. This method presents the advantage of being very simple and direct to implement. However, it can significantly decrease the runtime performance of SPH if too many additional particles need to be sampled, and approximating the geometry of the boundaries with particles can also introduce artifacts. To tackle these issues, implicit boundary approaches have been proposed [15]. They do not model the boundary explicitly like previously but implicitly, using signed distance fields [47] or regular grids [79]. Nevertheless, they require a pre-processing step that can be very time consuming in the case of high-resolution grids.

Since SPH methods enable the simulation of many materials, the animation of multi-phase fluids is quite straightforward in this framework. However, some precautions must be taken when handling the interface between different types of materials. For example, if some physical properties, such as rest density, are too different between the phases, it can create undesirable behaviors and discontinuities. Several works have proposed to tackle this issue, mainly by introducing a new density calculation [56, 136]. In this formulation, the density of a particle is calculated as if all of its neighbors had the same physical properties as it has, so that only the geometrical information is taken into account. This novel density can then be used in the forces calculation, and it prevents the discontinuities that were observed otherwise.

Finally, Macklin and Müller [92] proposed to adapt the position-based dynamics method (PBD) – initially presented for solid body simulation by Müller and colleagues [102] – to fluid animation. In PBD methods, the constraints are applied directly to the positions, avoiding the more typical manipulation of velocities, thereby making the results more controllable. This position-based fluids technique allows for a more stable simulation of incompressible flows than some traditional SPH solvers, enabling real-time applications with larger time steps.

For more details on important advances in SPH-based techniques, we refer the reader to the state-of-the-art reports of Ihmsen and colleagues [61] and Koschier and colleagues [80]. They detail contributions on data structures, boundary handling, pressure solve, viscous forces and many other points.

3.1.3 Hybrid methods

Finally, some hybrid methods have been proposed, using either an Eulerian or a Lagrangian representation during the different stages of the solving. The first hybrid solver was introduced in 1962 by Harlow [48] as the particle-in-cell method (PIC). In this paper, he proposed to store the quantities on particles that are distributed all over the fluid domain. The steps that are not related to advection are then processed on a grid, first requiring the transfer of the necessary quantities to the grid. Once these steps are performed on the grid, as they would be with any typical Eulerian solver, the physical quantities are transferred back to the particles thanks to a simple interpolation, and the advection step is performed within the Lagrangian specification. One major drawback of this method is that the consecutive interpolations needed to go from one representation to the other tend to smooth out the results.

Twenty-five years later, Brackbill and Ruppel introduced the fluid implicit-particle method (FLIP) [20, 21]. This paper was based on the PIC method, but instead of interpolating the entire set of physical quantities from the grid back to the particles, they only interpolated the *change* that had been computed on the grid. This proved to significantly reduce the numerical diffusion that could be observed within the PIC method. Since these two methods were used for compressible flows, Zhu and Bridson proposed in 2005 to adapt FLIP for incompressible flows [166]. Moreover, Raveendran and colleagues proposed in [115] to solve for a divergence-free velocity on a coarse grid and then transfer the pressure values on particles. The typical SPH steps were then performed on the particles, including the density correction from previous work.

Finally, the material point method (MPM) has also been used as a hybrid technique to animate fluids as well as materials that have fluid-like behaviors (such as sand or snow), or to simulate the interactions between both [142, 144]. It relies on a Lagrangian representation, combined with a background mesh or grid, yet it is considered as a "meshless" technique. Indeed, contrary to PIC and FLIP, the material points in MPM are equipped with strain and stress on top of mass, position, velocity, etc.

3.2 Deep learning

Artificial intelligence (AI) is a very active field, with new topics arising at a pace rarely seen in computer science research. This field relies on the knowledge of a given set of data, which is used to extract information in order to fulfill a given objective. *Machine learning* [19, 103] is a sub-field of AI in which algorithms learn to find correlations and patterns in the features of the data they are presented with. From these patterns, the systems can make predictions and extrapolate to unseen data. Finally, *deep learning* is a branch of machine learning that shares the same goals but uses different tools – described in Chapter 2 – to achieve them. In the following sections, we present the most important and game-changing contributions that were made with deep learning since the years 1950, and then detail the advances in the hybrid field of *physics-based deep learning*, which is the focus of our first contribution. For a complete survey and detailed technical explanation of deep learning techniques, we recommend reading the book of Goodfellow and colleagues [45].

3.2.1 Deep learning methods

Deep learning is based on the training of *neural network models*, which are composed of stacked layers containing nodes that are called neurons. These models are trained on large amounts of data in order to learn a specific task, such as classification, segmentation, or other

Figure 3.3: Results from (left) the first StyleGAN model [65] and from (right) "Denoising Diffusion Probabilistic Models" [53], showing images entirely generated by deep neural networks.

types of predictions. Neural networks were first introduced by Rosenblatt in 1958 [121], who found inspiration in the human brain for their design. These first networks only got three layers, but they were the first to enable the classification of data that was not linearly separable. Most of the models that were proposed after that contained more than three layers of neurons, and were thus called *deep neural networks* (DNN). Stochastic gradient descent was introduced for the first time in 1967 [4] as a technique to optimize these DNNs. Later, in 2014, the ADAM optimizer was proposed by Kingma and Ba [75] and has been used extensively ever since.

Deep neural networks have been introduced to various fields since the years 2000, such as natural language processing (NLP), image or video restoration, and image generation. For instance, the *multi-layer perceptron* (MLP) of Bengio and colleagues overpeformed the state-of-the-art model in NLP in 2003 [16], and many others have improved the processing of language since then [44]. Moreover, *convolutional neural networks* (CNN) were introduced for image processing in 1989 by Le Cun and colleagues [83], being lighter and better suited for large data than MLPs. Some very famous deep CNNs were inspired by this work and presented between 2012 and 2018, showing outstanding performance for image recognition, such as AlexNet [81], ResNet [51], VGG [134] and DenseNet [58].

Image segmentation has also been developing with the rise of deep neural networks, aiming at finding the boundary of the objects present in an image. Segmentation is commonly used in biomedical imaging, for example for tumor detection. The well-known *U-Net* architecture [120], introducing skip-connections in encoder-decoder models, was first proposed for brain tumor detection in 2015 and was then widely used for other applications.

Furthermore, the now famous *attention mechanism* was first introduced in natural language processing. It started from the observation that, in NLP, one has to consider more than just the direct neighbors of a word, and that every word does not have the same importance for the meaning of a sentence. In 2017, Vaswani and colleagues presented their *transformer* architecture [155] for language processing tasks, that was leveraging this attention mechanism. Transformers were then successfully adapted to computer vision in 2021 [39].

Content synthesis has been at the heart of deep learning research since 2014, experiencing a significant acceleration in 2020 with the rise of diffusion models. Goodfellow and colleagues introduced the *generative adversarial network* (GAN) architecture in 2014 [46]. A GAN is a model composed of a generative and a discriminative network, which are competing against each other. The former is trying to create content that is as realistic as possible in order to fool the latter, whose goal is to find whether this content has been artificially generated or not. Many contributions have been made in image synthesis since the first GAN was proposed, such as the CycleGAN model [165] or the very popular StyleGAN architecture [65, 67, 66] (Figure 3.3

- left). *Variational autoencoders* (VAE) have also shown excellent performance [76], allowing for control over their latent space probabilistic distribution. Nevertheless, the quality of the generation is usually poorer than with GANs, although synthesizing a better diversity of samples. Moreover, *diffusion models* have been introduced for image synthesis in 2020 by Ho and colleagues [53]. Their work (Figure 3.3 - right), that was inspired by diffusion probabilistic models for thermodynamics, gave high quality results in terms of fidelity and diversity, with improved controllability compared to GANs. However, such models are much heavier and slower to train than GANs or VAEs. Rombach and colleagues tackled this issue in [119], where they let diffusion models operate in the latent space of pre-trained autoencoders instead of applying them in pixel-space directly. Finally, text-to-image models have known unprecedented advances in 2022, with the successive publications of DALL-E 2 by OpenAI and Imagen by Google Research [114, 122]. The former was based on the CLIP algorithm [113, 108] from the NLP field, whereas the latter used transformer networks. The public quickly gained access to simplified user interfaces implementing these algorithms, making them a very popular subject of discussion and debate. During the same year, OpenAI released ChatGPT – a chatbot that has become extremely popular – that was based on generative pre-trained transformers (GPT), which are a type of large language models.

3.2.2 Physics-based deep learning

In this thesis, we focus on deep learning algorithms used in conjunction with physics-based modeling, in particular for the animation of fluid flows. Lots of research topics combining machine learning and numerical solvers have arisen with the very fast development of the former [33, 69, 24], and we propose to introduce the most important ones in this section. For a more thorough overview of physics-based deep learning, we recommend the book published by Thuerey and colleagues in 2021 [146], as well as the review published by Karniadikis and colleagues during the same year [64].

Machine learning algorithms highly depend on the data they are trained on, and collecting correct and meaningful data is a crucial step to ensure the quality of the models. Fluid solvers can thus be used to produce data that will be processed by deep learning models, in order to predict accurate fluid simulations. Indeed, if a deep learning model is trained with physically realistic data output by a trusted numerical solver, then this model has more chance to be able to predict physically realistic solutions. Therefore, a conventional direction when using machine learning for PDEs has been to aim for the replacement of entire PDE solvers by neural network models that can efficiently approximate the solutions [91, 72, 157, 18]. In this context, Fourier Neural Operator [86] and Neural Message Passing [22] models have also been introduced for learning PDEs, aiming at a better representation of full solvers with neural network models. Furthermore, for smoke simulations in particular, some works focused on super-resolution models proposing efficient deep neural network approaches that synthesized high-resolution results from low-resolution versions [30, 161, 41, 10], or that converted low frame rate results into high frame rate versions [107].

Moreover, numerical solvers and deep learning frameworks can be used in an interleaved manner, each influencing the other in the learning and solving processes. In such hybrid methods, the data is produced by a solver, it is used to train a deep neural network model for a specific task, and the data inferred by this model can in turn be used by the solver. For example, a learned model can replace the most expensive part of an iterative PDE solver for fluids [147, 160] or supplement inexpensive yet under-resolved simulations [150, 135] (Figure 3.4). In such methods, the numerical solver needs to be differentiable, in order to be integrated into the training pipeline of deep learning models. Since the years 2010, differentiable components

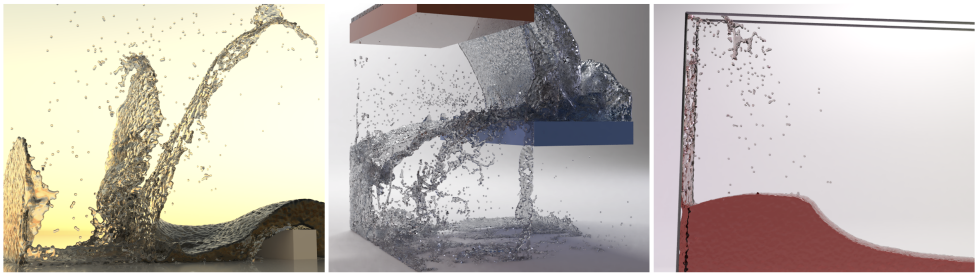

Figure 3.4: Results from the "Liquid Splash Modeling with Neural Networks" [150]. The small-scale splashes are generated using a neural network model.

for machine learning have been studied extensively, particularly when training neural network models in recurrent setups for spatio-temporal problems [5, 8, 148, 28, 125, 87, 158, 151, 78, 167]. Consequently, a variety of differentiable programming frameworks have been developed for different domains [126, 57, 62, 54]. These differentiable frameworks allow neural networks to closely interact with PDE solvers, which provides the model with important feedback about the temporal evolution of the target problem.

Finally, particle-based simulations have been at the heart of numerous research works combining deep learning with physical modeling. Ladicky and colleagues first proposed to use regression forests as a means to predict the states of liquid simulations [82], which showed the advantage of handling incompressibility as well as the addition of external forces at inference time only. Tumanov and colleagues proposed in [149] to use a 3D convolutional neural network that aimed at replacing the incompressibility solver from the "Position-based fluids" paper [92], by first translating the particle and obstacle data into a grid structure to give as input to the network. Both methods allowed for the simulation of millions of particles in real-time. On the other hand, Ummenhofer and colleagues applied CNNs directly to the particle data, enabling an improved accuracy over the particles' trajectories [152]. Several works also relied on graph neural networks to simulate particle-based fluids [123, 124, 85], considering each particle as a node and simulating their interactions through the graph edges.

3.3 Intrinsic geometry processing

In this section, we do not aim at reviewing the entire geometry processing literature, since this vast field comprises many branches that are not directly related to our contributions. Instead, we focus on the works tackling the computation of geodesics and parallel transport of vectors, which are the main focus of this thesis in terms of geometry processing techniques.

In Chapter 1, we defined the difference between *intrinsic* and *extrinsic* mesh representations. The former proved particularly well-suited when it came to building robust algorithms, the latter often leading to results of poor quality with real-world, low-quality meshes. Indeed, extrinsic algorithms are generally designed on artificial, ideally constructed meshes, whereas in reality most of them have unperfect triangulations. Moreover, intrinsic representations facilitate working with non-orientable meshes. Therefore, several works focused on proposing intrinsic representations or computations for more robustness. For instance, Sharp and colleagues as well as Gillespie and colleagues proposed methods to create *intrinsic triangulations* for poor-quality meshes without changing their geometry, making the application of algorithms that require high-quality meshes possible [130, 43] (see Figure 3.5). In this context, it is

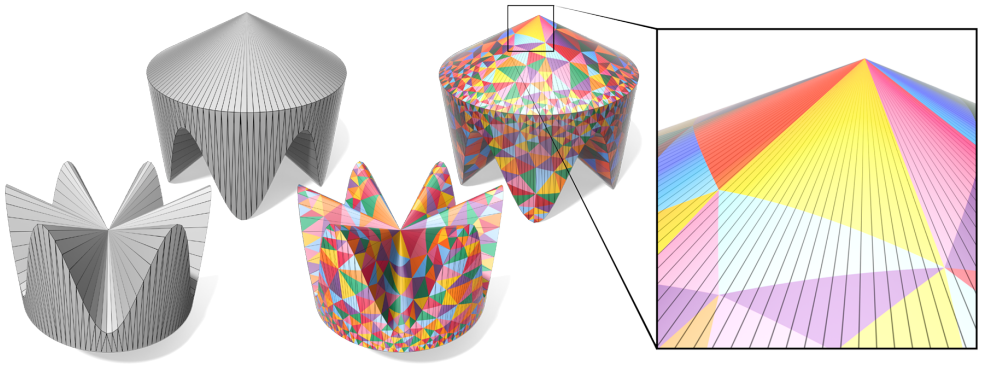

Figure 3.5: Intrinsic triangulation from "Navigating Intrinsic Triangulations" [130], in color. The original, low-quality triangulation is shown in grey.

important to be able to compute geodesic paths and distances on polyhedral meshes, which usually replace the Euclidean metric in intrinsic approaches.

In Chapter 2, we introduced the algorithm of Mitchell, Mount and Papadimitriou (MMP) [95] to compute exact geodesics by propagating *windows* on polyhedral meshes. This algorithm as well as the one proposed by Chen and Han three years later in 1990 [27] were computationally intensive and thus needed the introduction of acceleration structures to be applied [143]. Crane and colleagues introduced the *heat method* for geodesic computation in 2013, enabling the computation of geodesics on regular grids, meshes as well as point clouds by solving sparse linear systems [32]. Nonetheless, their method resulted in approximations that could be very different from exact geodesics. Sharp and Crane proposed a simple and efficient method to find geodesic curves on polyhedral meshes, by flipping edges of the mesh iteratively [129]. Their work however was not meant to compute geodesic *distances*. For a complete survey of methods that enabled the computation of geodesic paths and distances, we recommend the work of Crane and colleagues [68].

Once geodesic paths are defined, one can use *parallel transport* to transport vectors along them, by ensuring that the vectors stay parallel to the curve in the manifold. We focus on *discrete* parallel transport, considering that a smooth surface can be discretized into locally tangent planes. In order to use parallel transport on such a surface, it needs to be equipped with *connections* that describe the mapping between these tangent planes. In 2010, Crane and colleagues proposed a method to build connections on discrete surfaces [31], which proved to be efficient and straighforward since it was only using standard operators. Sharp and colleagues proposed in 2019 "The vector heat method" [131], building on the works of Polthier and Schmies [110] and Knöppel and colleagues [77]. With this method, they allowed for the computation of parallel transport of general vector-valued data on a curved manifold. To do so, they did not build typical geodesic paths, but instead solved linear systems thanks to discrete Laplacians and to the "vector heat equation", simulating a diffusion process over the manifold.

Chapter 4

Exploring physical latent spaces

We introduced the necessary technical notions and state-of-the-art works in the domains of fluid simulation, deep learning and geometry processing. We now present our first contribution, which aims at exploring physical latent spaces of deep neural networks for the simulation of turbulent fluids. This dimension reduction enables the acceleration of typical solvers while improving the results from previous works.

This work has been presented as ***Exploring Physical Latent Spaces for High-Resolution Flow Restoration,*** *Chloé Paliard, Nils Thuerey and Kiwon Um, in the proceedings of the 28th International Symposium on Vision, Modeling, and Visualization (VMV), September 27-29, 2023, Braunschweig, Germany.*

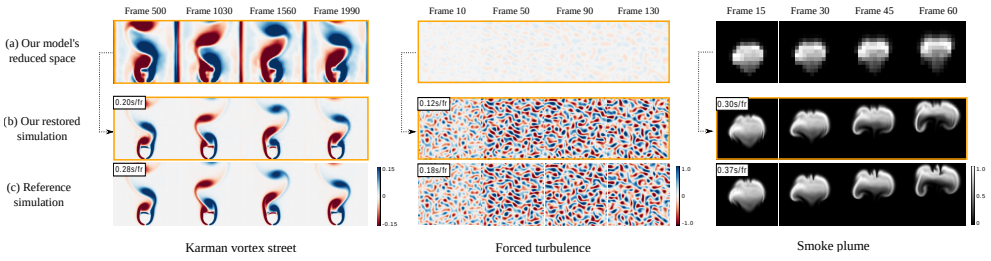

Figure 4.1: We propose a model composed of neural components and a physics solver, that autonomously discovers the reduced representation (a) that best fulfills the goal of restoring a fine reference simulation (b, c) from a unique coarse frame. This leads to relative improvements with respect to the baseline of 91% on average for the Karman vortex street case, 74% for the forced turbulence case and 35% for the smoke plume case.

4.1 Introduction

In this chapter, we present a data-driven simulation technique for PDE problems with a novel training method that explores using physical states as latent space for deep learning. In contrast to many previous studies [100, 72, 97, 138], our latent space is not composed of the output or intermediate states of a neural network, but is rather made of the physical states of a PDE solver, such as velocity fields. We train a deep neural network (DNN) to exploit the content of a reduced PDE solver and shape it in a way that best satisfies the given learning objective, i.e., achieving solutions that are as accurate as possible at our target high resolution space. This *shaping* of the physical latent space gives the neural network a chance to discover modified dynamics and allows our model to better restore accurate high resolution solutions from them. Examples of the reduced and restored solutions are shown in Fig 4.1.

Our training method consists of an *encoder* model transforming a coarse physical state using the degrees of freedom of a learnable latent space, a *physics solver* corresponding to a given PDE followed by an *adjustment* DNN, both operating in the reduced space, and a *decoder* turning the reduced state into the target high resolution space. To train our models with a physics solver, we adopt a differentiable simulator approach [57, 54, 146]. We let the encoder model learn the latent space representation without any other constraint than the restoration of the target solution. Therefore, an end-to-end training of this pipeline gives the encoder the complete autonomy to shape the reduced representation.

We demonstrate that the autonomy of our training method leads to a better performance than previous work, especially in terms of generalization. We apply our method to various complex, non-linear PDE problems, based on the Navier-Stokes equations, which are essential in the context of modeling fluid flows. For all the scenarios, our model produces more accurate high-resolution results in a longer temporal horizon than conventional and more tightly constrained models.

4.2 Related work

The important research works related to physics-based deep learning and differentiable solvers have been reviewed in Chapter 3, Section 3.2.2. In the following, we focus on previous works that specifically tackle latent space representations and reduced physical solutions.

Effectively utilizing latent spaces lies at the heart of many ML-based approaches for solving PDEs. A central role of the latent space is to embed important (often non-linear) information for the given training task into a set of reduced degrees of freedom. For example, with an autoencoder network architecture, the latent space can be used for discovering interpretable, low-dimensional dynamical models and their associated coordinates from high-dimensional data [26]. Moreover, thanks to their effectiveness in terms of embedding information and reducing the degrees of freedom, latent space solvers have been proposed for different problems such as advection-dominated systems [93] and fluid flows [159, 42]. While those studies typically focus on training equation-free evolution models, we focus on latent states that result from the interaction with a PDE solver. Neural network models have also been studied for the integration of a dynamical system with an ordinary differential equation (ODE) solver in the latent space [28]. This approach targets general neural network approximations with a simple physical model in the form of an ODE, whereas we focus on learning tasks for complex non-linear PDE systems.

The ability to learn underlying PDEs has allowed neural networks to improve reduced, approximate solutions. Residual correction models are trained to address numerical errors

of PDE solvers [151]. Details at sub-grid scales are improved via learning discretizations of PDEs [13] and learning solvers [78, 138] from high-resolution solutions. Moreover, multi-scale models with downsampled skip-connections have been used for super-resolution tasks of turbulent flows [41]. These methods, however, typically employ a constrained solution manifold for the reduced representation. Indeed, the reduced solutions are produced using coarse-grained simulations with standard numerical methods, while our work shows the advantages of autonomously exploring the latent space representation through our joint training methodology.

4.3 Exploring physical latent spaces

For a given learning objective, our training method explores how neural network models can leverage the physical states of a PDE as latent space. Let $\mathbf{f} \in \mathbb{R}^{d_f}$ and $\mathbf{r} \in \mathbb{R}^{d_r}$ denote two discretized solutions of a PDE, a fine and a coarse version respectively, with $d_r \ll d_f$. We focus on the numerical integration of this target PDE problem and indicate the temporal evolution of each state as a subscript. A reference solution trajectory integrated from a given initial state $\mathbf{f}_{t_0}$ at time t_0 for n steps is represented by the finite set of states $\{\mathbf{f}_{t_0}, \mathbf{f}_{t_1}, \cdots, \mathbf{f}_{t_n}\}$. Each reference state is integrated over time with a fixed time-step size using a numerical solver $\mathscr{P}_f$, i.e., $\mathbf{f}_{t+1} = \mathscr{P}_f(\mathbf{f}_t)$. Similarly, we integrate a reduced state $\mathbf{r}_t$ over time using a corresponding numerical solver $\mathscr{P}_r$ at the reduced space, which we will call *reduced solver* henceforth, i.e., $\mathbf{r}_{t+1} = \mathscr{P}_r(\mathbf{r}_t)$. In this chapter, we focus on cases where the solver $\mathscr{P}$ is the same for both reduced and fine discretizations.

Our model takes the bilinear down-sampling of $\mathbf{f}_{t_0}$, i.e., $\mathbf{s}_{t_0} = lerp(\mathbf{f}_{t_0})$, as input, and transforms it with the help of an encoder function $\mathscr{E}(\mathbf{s}|\theta_E) : \mathbb{R}^{d_r} \rightarrow \mathbb{R}^{d_r}$, thus $\mathscr{E}(\mathbf{s}_{t_0}|\theta_E) = \hat{\mathbf{s}}_{t_0}$. Then, we can obtain the next reduced state $\mathbf{r}_{t_1} = \mathscr{P}(\hat{\mathbf{s}}_{t_0})$. Moreover, in order to keep the reduced solution consistent with the encoded representation over time, the output of the reduced solver is transformed by an adjustment function, $\mathscr{A}(\mathbf{r}_{t_1}|\theta_A) = \hat{\mathbf{r}}_{t_1}$. Thus, each reduced state $\hat{\mathbf{r}}_{t_i}$ is obtained by i recurrent evaluations of the reduced solver and the adjustment function. Finally, a decoder function $\mathscr{D}(\mathbf{r}|\theta_D) : \mathbb{R}^{d_r} \rightarrow \mathbb{R}^{d_f}$ restores a fine solution trajectory $\{\hat{\mathbf{f}}_{t_0}, \hat{\mathbf{f}}_{t_1}, \cdots, \hat{\mathbf{f}}_{t_n}\}$ from the reduced trajectory $\{\hat{\mathbf{r}}_{t_0}, \hat{\mathbf{r}}_{t_1}, \cdots, \hat{\mathbf{r}}_{t_n}\}$, thus $\hat{\mathbf{f}}_{t_i} = \mathscr{D}(\hat{\mathbf{r}}_{t_i}|\theta_D)$. We model the encoder, adjustment, and decoder functions as DNNs in which trainable weights are denoted by θ_E, θ_A, and θ_D, respectively.

The joint learning objective of the three DNNs is to minimize the error between the approximate solutions and their corresponding reference solutions, i.e., $||\hat{\mathbf{f}}_t - \mathbf{f}_t||_2$. To guide the adjustment model, we additionally minimize $\|\hat{\mathbf{r}}_t - \mathscr{E}(\mathbf{s}_t|\theta_E)\|_2$. Thus, the final loss of our model is as follows:

$$\mathscr{L} = \sum_{i=1}^{N} \lambda_{hires} \times ||\hat{\mathbf{f}}_{t_i} - \mathbf{f}_{t_i}||_2 + \lambda_{latent} \times ||\hat{\mathbf{r}}_{t_i} - \mathscr{E}(\mathbf{s}_{t_i}|\theta_E)||_2 \tag{4.1}$$

where N denotes the number of integrated time-steps for training. Hence, at each training iteration, the gradients through all N steps are computed for back-propagation and, consequently, all the models get jointly updated.

Fig 4.2 shows the architecture of our approach. As the encoder does not receive any explicit constraint and has the complete freedom to *autonomously* explore the reduced space to arrive at a suitable representation, we denote this approach by *ATO*.

Comparisons and baselines To illustrate the capabilities of our physical latent space, we compare *ATO* to two state-of-the-art models that operate in coarse space [138, 151]. The

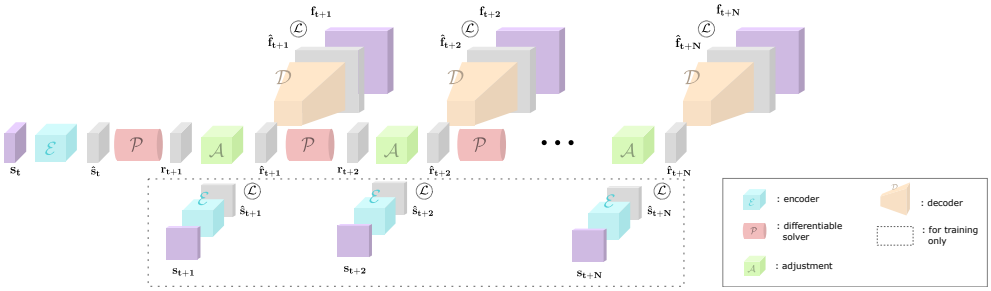

Figure 4.2: Architecture of our autonomous training approach for N integrated solver steps. The initial state is encoded into the latent space, the solver and adjustment models are applied N times, and the adjusted states are decoded into the fine space.

former, denoted by *Dil-ResNet* in the following, represents a neural network model that aims at directly predicting solution states in the reduced space at each time-step. Hence, it does not make use of the reduced physics solver $\mathscr{P}$. On the other hand, the work from [151], denoted by *SOL*, consists of a differentiable physics solver and a trainable corrector model that addresses numerical errors of the solution states. In both cases, unlike *ATO*, the models are trained to make reduced solutions by targeting the bilinear down-sampling of the reference.

We note that these state-of-the-art models output solutions that stay in the coarse space. As our *ATO* model aims at restoring high-resolution solutions using a decoder, a super-resolution model can transform these models' reduced solutions into high-resolution ones. To this end, we give the reduced states produced by the *Dil-ResNet* and *SOL* models to a super-resolution network specialized for spatio-temporal turbulence problems [41], resulting in high-resolution states. Henceforth, these models will be denoted as *Dil-ResNet + SR* and *SOL + SR*.

4.4 Experiments

In order to acquire a training data-set for each scenario, we generate a set of solution sequences of the given PDE problem. The PDEs from our experiments work with a continuous velocity field $\mathbf{v}$ in the two-dimensional space, i.e., $\mathbf{v} = [v_x, v_y]^T$. Considering reference simulations on regularly discretized grids, we focus on exploring latent spaces (i.e., reduced representations) that are four times coarser than the reference.

4.4.1 Karman vortex street

This first example targets a complex PDE problem within a constrained setup, where the velocity field evolves over time while being constrained to be divergence free. We evaluate the incompressible Navier-Stokes equations for Newtonian fluids:

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v}\cdot\nabla)\mathbf{v} - \frac{\nabla p}{\rho} + \nu\nabla^2\mathbf{v} \quad \text{subject to} \quad \nabla\cdot\mathbf{v} = 0 \tag{4.2}$$

where p is the pressure, ρ is the density, and ν is the kinematic viscosity coefficient. The reference simulation domain is discretized with 128×256 cells and a cell spacing of one using a staggered grid scheme. We use closed boundary conditions for the sides and open boundary conditions for the top of the domain; at the bottom, we set a constant inflow velocity. The continuous inflow collides with a fixed circular obstacle, which creates an unsteady wake flow

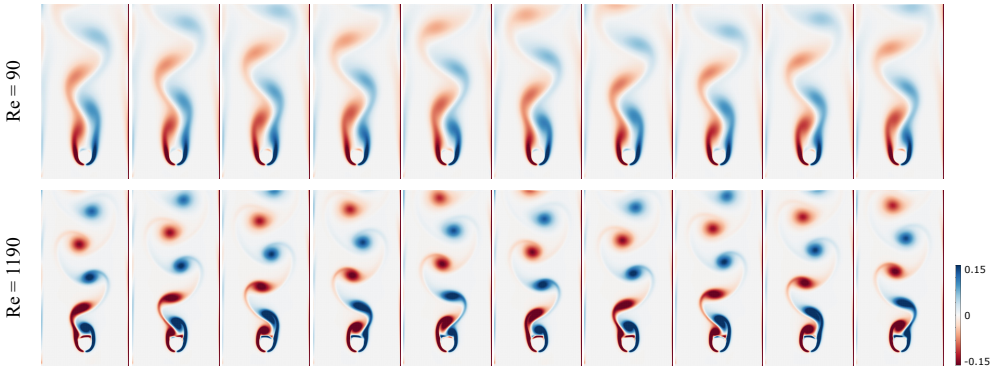

Figure 4.3: Two examples from the training data-set of the Karman vortex street scenario: Re = 90 (top) and Re = 1190 (bottom). The vorticity fields are shown.

that evolves differently depending on the Reynolds number. For the temporal discretization, a time step size of one is used.
We generate 20 simulations of 200 steps each, with the following Reynolds numbers: {90, 120, 140, 150, 160, 170, 180, 190, 200, 220, 290, 340, 390, 490, 540, 590, 690, 740, 790, 1190}. We first let the simulation run for 2000 time-steps, in order to let the flow stabilize. We randomly choose 5% of the generated frames for the validation set and the remaining 95% for the training set. Both the least and most turbulent simulations of the training set are shown in Fig 4.3.

In order to make our training more stable, we pre-train our networks with eight integrated steps as warm starts for our final models. Each training uses 100 epochs with a batch size of ten. The learning rate starts from 4×10^{-4} and exponentially decays with a decaying rate of 0.9 every ten epochs. If divergence happens while training, we restart our training with a smaller learning rate. In this example, we compare all the models trained with 16 integrated steps. The encoder and adjustment models of *ATO*, the corrector of *SOL*, and the solver of *Dil-ResNet* take the Reynolds number as additional input.

4.4.2 Decaying turbulence

This example tackles the same incompressible Navier-Stokes equations, but with vortices intialized all over the physical space that slowly decay over time. In this scenario, the viscosity stays constant (equal to 0.1) within the training and test data-sets. The reference simulation domain is discretized with 128^2 cells and a cell spacing of one. Both the discrete velocity and pressure values are stored at the center of each cell, and periodic boundary conditions are applied. For the temporal discretization, a time step size of 1.0 is used. The training data-set consists of 20 simulations of 200 steps each, which evolve from different initial velocity fields. We randomly select 5% of the frames for the validation set and the remaining 95% for the training set. An example sequence of the data is shown in Fig 4.4. As in the Karman vortex street case, we first train our models with eight integrated steps as warm starts for the final models. We compare all the models trained with 16 integrated steps.

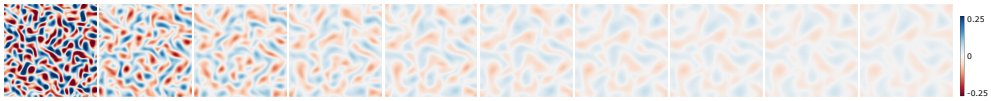

Figure 4.4: Example frames from one simulation of the training data-set of the decaying turbulence scenario. The vorticity fields are shown.

4.4.3 Forced turbulence

This case has the same experimental setup as the previous one, but with an external force sequence $\mathbf{g}(\mathbf{x},t)$ that is added to Eq. (4.2). This force sequence yields complex, chaotic evolutions of vortices over time. We use a different force sequence for each simulation trajectory, composed of 20 overlapping sine functions as follows:

$$\begin{aligned} g_x(\mathbf{x},t) &= \sum_{i=1}^{20} a_i \sin(k_i \alpha_i \cdot \mathbf{x} + w_i t + \phi_i) \\ g_y(\mathbf{x},t) &= \sum_{i=1}^{20} a_i \sin(k_i \alpha_i \cdot \mathbf{x} + w_i t + \phi_i) \end{aligned} \tag{4.3}$$

where a_i is the amplitude, k_i is the wave number, α_i is the wave direction, w_i is the frequency, and ϕ_i is the phase shift. These values are randomly sampled from uniform distributions as follows: $a_i \in [-0.1, 0.1]$, $k_i \in \{6, 8, 10, 12\}$, $w_i \in [-0.2, 0.2]$, and $\phi_i \in [0, \pi]$. α_i is a random angle ($\in [0, 2\pi]$). The composed sine functions are, then, evaluated over the domain mapped into $[0, 2\pi]$ for each dimension.

For the temporal discretization, a time step size of 0.2 is used. The training data-set consists of 20 simulations of 200 steps each, which evolve from different initial velocity fields with different force sequences. We randomly select 5% of the frames for the validation set and the remaining 95% for the training set. An example sequence of the data is shown in Fig 4.5. We use the models trained on the previous decaying turbulence case as warm starts for our final models, trained for 100 epochs. We compare all the models trained with 16 integrated steps.

In this set-up, a reduced force field needs to be applied at each time-step to the reduced velocity field by the solver. Thus, the encoder of *ATO* shares its weights for velocity and force in order to learn a unified operation for both reduced representations. At each time-step t, the linearly down-sampled version of the force field $lerp(\mathbf{g}(\mathbf{x},t))$ is passed to the *encoder* model, and the output is used for the reduced solver step. The adapted architecture for this set-up with external forces is shown in Fig 4.6.

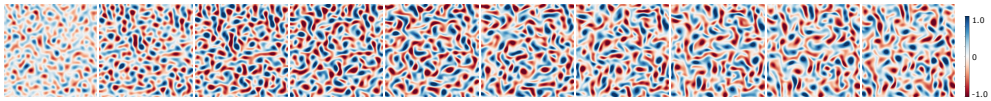

Figure 4.5: Example frames from one simulation of the training data-set of the forced turbulence scenario. The vorticity fields are shown.

4.4.4 Smoke plume

This last scenario serves as a proof-of-concept of our method for more complex, practical graphics applications. Aiming for complex flow behavior driven by hot smoke plumes, we

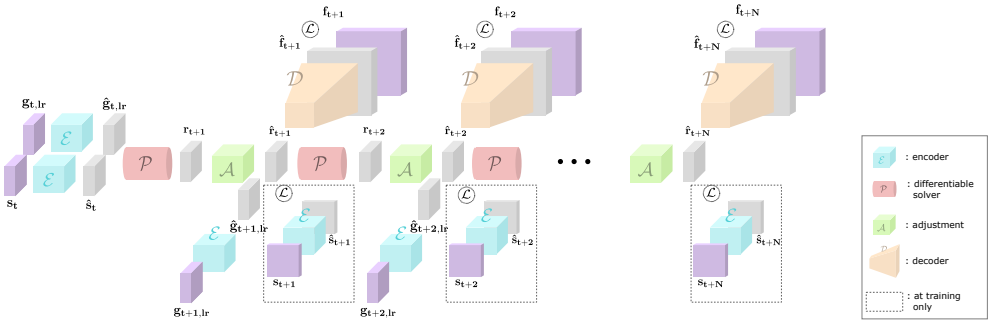

Figure 4.6: Architecture of our autonomous training approach for N integrated solver steps in the case where external forces $\mathbf{g_t}$ are used.

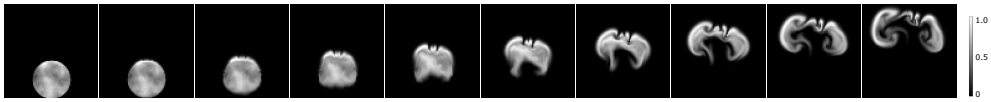

Figure 4.7: Example frames from one simulation of the training data-set of the smoke plume scenario. The marker fields are shown.

set up an initial smoke volume as a marker field with an arbitrary density distribution in a circular shape. The marker field is then passively advected by the velocity field and, at the same time, induces a buoyancy force via the Boussinesq approximation, that is influencing the velocity evolution. Therefore, the marker and velocity fields are tightly coupled. This scenario considers a more challenging problem of the Navier-Stokes equations than before, naturally making the fluid flow more interesting and providing a harder task for our model. The simulation domain is discretized with 128^2 cells adopting a centered layout for the marker field, a staggered layout for the velocity field, and open boundary conditions. The passive marker field is given as an input to our encoder and adjustment models, but it is only used as additional information. Thus, the linearly down-sampled version of this marker field is used in the reduced solver. Then, only the velocity field is up-sampled, and the high-resolution reference marker field is advected by the predicted velocity.

The training and test data-sets are composed of smoke volumes initialized with random noise, with a fixed position and radius (of 0.12). We use randomly selected 5% for the validation set and the remaining 95% for the training set. An example sequence of the data is shown in Fig 4.7. For this case, we use more integrated solver steps than the others. We first train our model with four, eight, and 16 integrated steps as warm starts for the final model. Finally, we apply our *ATO* model trained with 32 steps, for 100 epochs.

4.4.5 Network architecture and training procedure

In this section, we detail the network architectures that compose our *ATO* model (Fig 4.8), along with the state-of-the-art networks that we compare it to.

The encoder of the *ATO* setup consists of two convolutional layers with 32 and 16 features each with a kernel size of five. Each convolutional layer is followed by the Leaky ReLU activation function. A last layer with two features and the same kernel size but without the activation infers the final encoded output. This model has approximately 15k trainable weights.

The adjustment of the *ATO* setup and the corrector of *SOL* ([151]) employ an identical

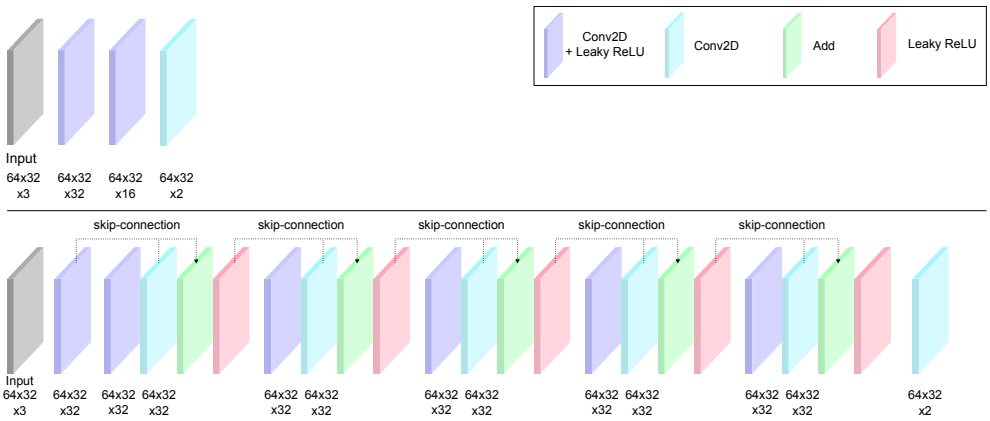

Figure 4.8: Architecture of the encoder (top) and adjustment (bottom) networks of our *ATO* model.

network model. This model consists of a first convolutional layer with 32 features and a kernel size of five, followed by five blocks of two convolutional layers with 32 features each and a kernel size of five. Each layer is followed by the Leaky ReLU activation function, and each block is connected to the next with a skip-connection. A last layer with two features follows with the same kernel size yet without the activation. This architecture has approximately 260k trainable weights.

The decoder used for the *ATO* setup and the super-resolution model from *Dil-ResNet + SR* and *SOL + SR* are adapted from the multi-scale architecture of [41], such that the total number of trainable weights is close to 97k.

The *Dil-ResNet* model is adapted from the architecture of the state-of-the-art network model proposed for turbulent flow problems [138]. This model has a first convolutional layer with 32 features with a kernel size of three and no activation. It is followed by four identical blocks of seven convolutional layers with 32 features each with a kernel size of three and varying dilation rates from one to eight (respectively: 1, 2, 4, 8, 4, 2, 1). Each layer is followed by the ReLU activation function, and each block is linked to the next via a skip-connection. A last layer with two features follows with the same kernel size yet without the activation. This model has a similar number of trainable weights as the adjustment model's (i.e., 260k). We note that a larger model did not improve the performance. Contrary to the other models, *Dil-ResNet* is trained for only one step at a time and uses a *MSE loss*.

For the Karman vortex street and smoke plume cases, we adopt zero-padding, and for the forced and decaying turbulence cases, which use periodic boundary conditions, we use periodic padding for all models. At each training iteration, for a given batch size, we randomly sample the initial states from the reference solution trajectories and integrate the approximate solution trajectories for N steps. All our trainings use an Adam optimizer [75] and a decaying learning rate scheduling.

Training hyper-parameters Firstly, the code of the *SOL* model from [151] was released publicly, which enabled an easy reproduction of their experiments. Secondly, the learning setup of the *Dil-ResNet* model from [138] was precisely described in the article. We extensively tested the various hyper-parameters and chose the *Dil-ResNet* model over *Con-Dil-ResNet* (i.e., without the additional loss constraint) because it performed better in our physical scenarios. For training, we chose a Gaussian noise with $\sigma = 0.01$ for all scenarios.

Lastly, for our *ATO* setup, we chose the depth of the models by making a compromise between performance and runtime/resources. For the learning rate and batch size, since the physics solver made the models harder to train, we chose the values that best stabilized our training. Our loss being divided in two terms of different orders of magnitude (a high-resolution term and a low-resolution one), we set λ_{hires} to 1 for all scenarios and $\lambda_{latent} = 1$ for the Karman vortex street case, $\lambda_{latent} = 1$ for the decaying turbulence, $\lambda_{latent} = 100$ for the forced turbulence and $\lambda_{latent} = 10$ for the smoke plume.

4.5 Results

We evaluate the trained models based on relative improvements over a *baseline* simulation. To build the baseline solutions, we simply apply the solver to the linearly down-sampled frames, and up-sample the reduced states into the reference space with a bilinear interpolation. Errors of the different restored solutions $\hat{\mathbf{f}}_t$ are computed with respect to the reference solutions $\mathbf{f}_t$, and the improvement over the baseline is calculated as follows:

$$improvement = \frac{\|\mathbf{baseline}_t - \mathbf{f}_t\| - \|\hat{\mathbf{f}}_t - \mathbf{f}_t\|}{\|\mathbf{baseline}_t - \mathbf{f}_t\|} \tag{4.4}$$

hence an improvement of 100% would mean that the restored solutions are identical to the reference. We evaluate each model using the mean absolute error (MAE) and mean squared error (MSE) metrics, which we measure in both velocity and vorticity. We present the results of the models trained with the highest number of integrated steps for each scenario, as they show better performance in general.

4.5.1 Reduced representations

The images of Fig 4.9 show visual examples of the reduced representations for the Karman vortex street and forced turbulence scenarios, for different time-steps. The graphs of Fig 4.10 show the quantified differences between the reduced states produced by the different trained models and the conventionally down-sampled reference states. We observe that our training procedure leads the latent representation to have vortex structures that are very similar to conventional down-sampling, while being considerably different quantitatively. For example,

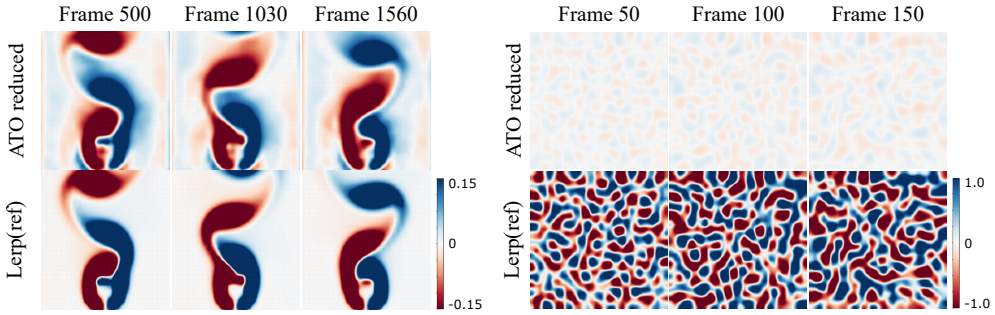

Figure 4.9: Reduced frames for the Karman vortex street case (left), and the forced turbulence case (right). The latent vorticity fields of *ATO* (top) and the linearly down-sampled reference (bottom) are shown.

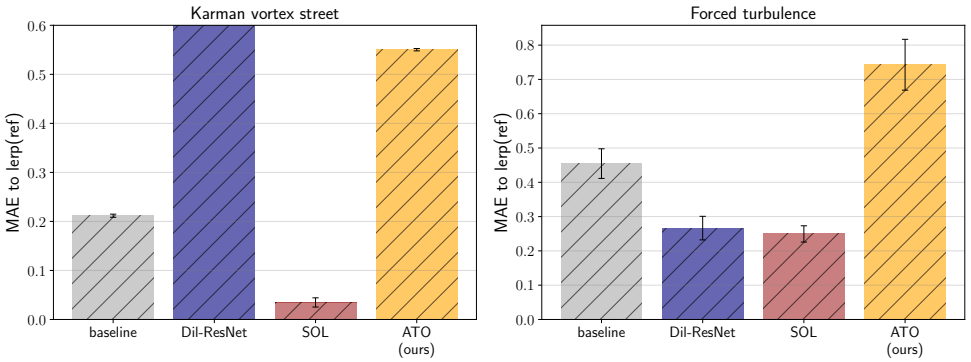

Figure 4.10: Distance between each model's reduced space and the down-sampled reference. The error bars indicate the standard deviation over the test runs.

one can notice on Fig 4.9 (right) that the vortex structures in the forced turbulence case are very similar but the colors are inverted and much lighter for our reduced latent space, which means that the velocity values are much lower in our case. We believe that the reduced representation of the *ATO* model stays physically meaningful for the numerical solver yet changes the content of the frames for accurately decoding high resolution states. We note that different training initializations of the same scenario produce latent representations that stay close to each other, which indicates that there exists a manifold of latent solutions that our *ATO* model converges to in order to get the best performance.

4.5.2 Karman vortex street

This example considers different vortex shedding behaviors depending on the Reynolds number of each simulation. We evaluate the models trained with 16 integrated steps on six test simulations with Reynolds numbers in {450, 650, 850, 1050, 1200, 1400}, consisting of 2000 time-steps each. In this scenario, we test the extrapolation ability of the models both physically and temporally, with higher Reynolds number thus more turbulent simulations than for training, and ten times longer sequences.

Table 4.1, along with Fig 4.11, which shows the velocity and vorticity error improvements of each model over the baseline, shows that *ATO* outperforms the other models, with an average relative improvement of 91% (and 88%) in terms of velocity MAE (respectively vorticity), while *SOL + SR* improves the baseline by 84% (and 83%) on average. On the other hand, the *Dil-ResNet + SR* model fails to retrieve the target simulation for more than 200 time-steps, and thus seems incapable of generalization in this scenario. The temporal metrics shown in Fig 4.12 demonstrate the capability of *ATO* to correctly restore a solution for longer time ranges than the other models. More specifically, the distance between the reduced states and *lerp (ref)* show that the *ATO* model is the only one to have a consistent latent representation over time, which proves its better temporal extrapolation capabilities.

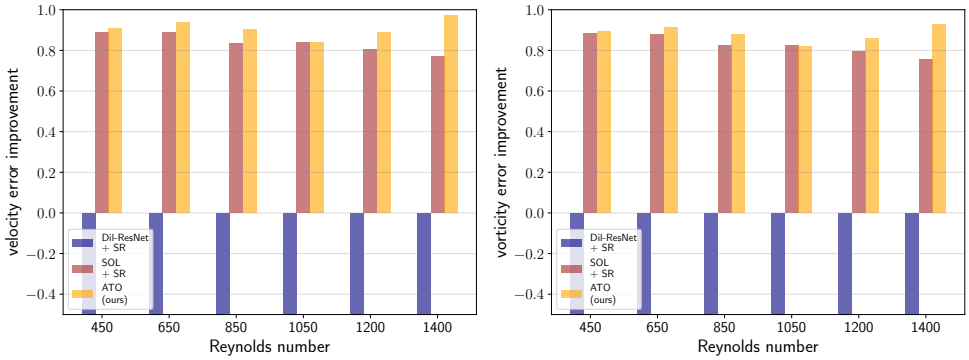

Figure 4.11: Velocity (left) and vorticity (right) error improvements (the higher the better) for six different Reynolds numbers between 450 and 1400. The highest Reynolds number used for training is 1190. The *ATO* model generalizes better than the others.

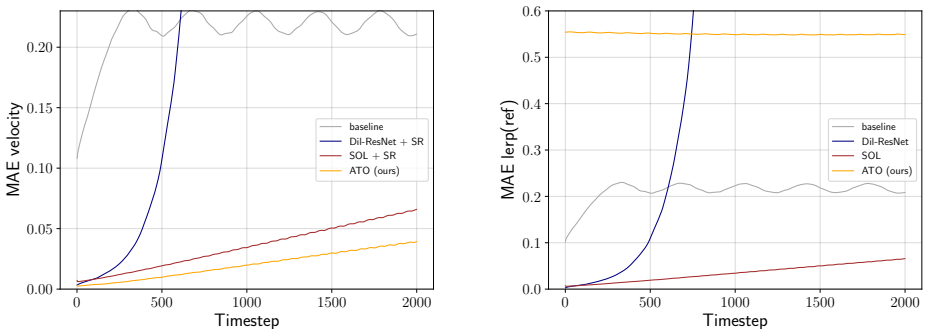

Figure 4.12: MAEs of recovered velocities (the lower the better) (left) and distances of the reduced spaces to the down-sampled reference (right) over time for the Karman vortex street scenario.

4.5.3 Decaying turbulence

In this example, we consider initially chaotic turbulent flows that slowly decay over time. We evaluate the models trained with 16 integrated steps, on five random initializations, lasting 200 steps each.

Table. 4.1 and Fig 4.13 show that the *ATO* model yields greatly improved results with a relative improvement of 83% (and 80%) in terms of velocity MAE (resp. vorticity) on average. However, in this more simple case, the *SOL + SR* model also improves the baseline significantly with 82% (and 78%) of average relative improvement. *Dil-ResNet + SR*, however, only yields 53% (and 6%) of improvement. As shown in Fig 4.14, *ATO*'s latent space representation is more distant from the linearly down-sampled representation than the other models', yet it shows similar or better performance.

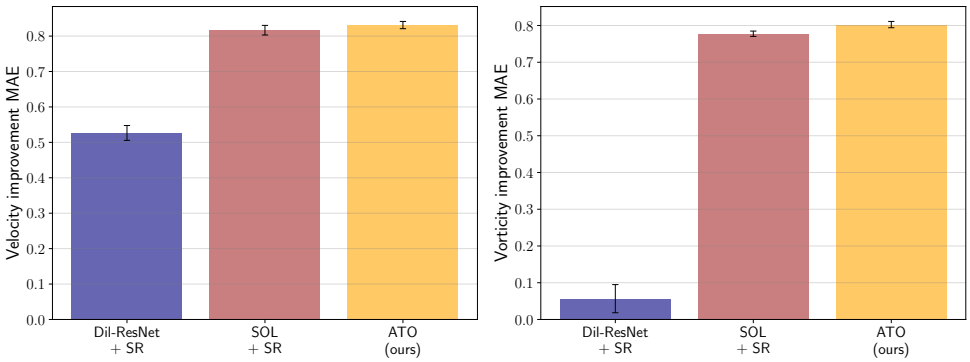

Figure 4.13: Velocity (left) and vorticity (right) error improvements (the higher the better) for the decaying turbulence scenario. The *ATO* model improves the baseline the most for every test case.

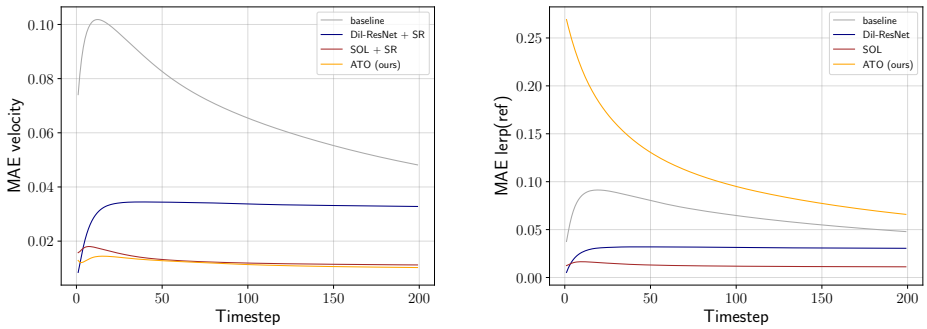

Figure 4.14: MAEs of recovered velocities (the lower the better) (left) and distances of the reduced spaces to the down-sampled reference (right) over time for the decaying turbulence scenario.

4.5.4 Forced turbulence

This complex fluid flow scenario considers the same experimental setup as in the previous case but with external forces, which leads to highly chaotic turbulent flows. We evaluate the models trained with 16 integrated steps, on five random initializations both in velocity and forcing, for 200 steps.

Table 4.1 and Fig 4.15 show that the *ATO* model significantly improves the baseline with a relative improvement of 74% (and 69%) on average in terms of velocity MAE (resp. vorticity). In comparison, *SOL + SR* improves by only 49% (and 43%) on average and *Dil-ResNet + SR* by 46% (and 38%). Therefore, in this complex case with external forcing and more turbulent flows, the *ATO* model particularly stands out, while its latent space representation (Fig 4.16) is once again more distant from the linearly down-sampled representation than the other models'.

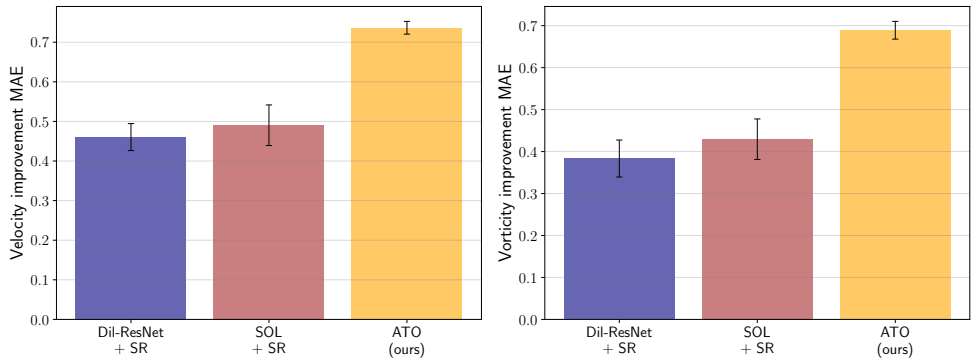

Figure 4.15: Velocity (left) and vorticity (right) error improvements (the higher the better) for the forced turbulence scenario. The *ATO* model improves the baseline the most for every test case.

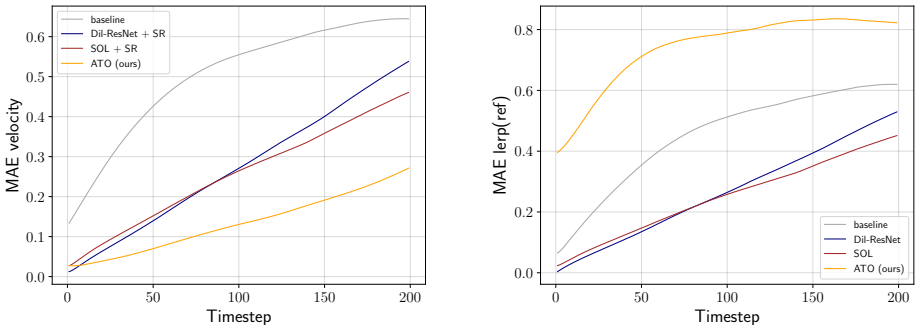

Figure 4.16: MAEs of recovered velocities (the lower the better) (left) and distances of the reduced spaces to the down-sampled reference (right) over time for the forced turbulence scenario.

	Karman vortex street (128×256)			Decaying turbulence (128×128)			Forced turbulence (128×128)		
	MAE	MSE	runtime	MAE	MSE	runtime	MAE	MSE	runtime
Reference	N/A	N/A	28.2	N/A	N/A	14.1	N/A	N/A	17.9
Baseline	0.214	0.096	11.1	0.069	0.024	7.7	0.504	0.426	9.6
Dil-ResNet+SR	3580	1407	9.2	0.033	0.023	3.8	0.272	0.167	5.3
SOL+SR	0.035	0.005	20.0	0.013	0.005	12.8	0.256	0.137	14.2
ATO (ours)	**0.020**	**0.002**	20.4	**0.012**	**0.005**	11.5	**0.133**	**0.040**	12.4

Table 4.1: Summary of the MAE and MSE metrics, along with the runtime for one simulation of 100 frames (averaged over ten runs, in seconds).

4.5.5 Ablation study

In order to see the effects of each of its components, we evaluate our *ATO* model with differently ablated training setups for the forced turbulence scenario. Our ablation study includes the following models:

- No latent loss: we remove the second term of the loss in Eq. 4.1; consequently, our training does not constrain the adjusted states to match the encoder-induced latent space.
- No encoder: we omit the encoder such that the latent representation is constrained to be conventional bilinear down-sampling.
- No encoder & no latent loss: since the previous model's reduced space is constrained to bilinear down-sampling, we test the same setup without the encoder and with no latent constraint.
- No solver: we replace the *solver + adjustment* part of our *ATO* model with the *DilResNet* NN-solver in order to study the effect of a non-physical latent space.
- No adjustment: we evaluate a setup where the reduced simulation evolves without being adjusted.
- *lerp (forces)*: we input a simple bilinear down-sampling of the force fields to the reduced solver, instead of their encoded representation.

Table 4.2 and Fig 4.17 show that the encoder, physics solver, and adjustment components of our *ATO* model are essential for its good performance. Firstly, the *no encoder* and *no encoder & no latent loss* experiments confirm that, with *lerp (ref)* as initial reduced representation and without our encoder, the adjustment network was not able to find a latent representation that would lead to an optimal performance. Furthermore, the *no latent loss* ablation shows that the latent loss guiding the adjustment model via the encoder results in a better performance. Note that the performance of *ATO* significantly decreased when the encoder was absent, whereas the performance drop due to omitting the latent loss was relatively less significant. Secondly, the *no solver* and *no adjustment* experiments show that using a reduced physics solver in conjunction with an adjustment model is crucial for the good performance of our *ATO* model. Finally, the *lerp (forces)* experiment indicates that our encoder model failed to find a latent representation for the velocity that was compatible with an external factor conditioned to *lerp*.

All of the models tested in this ablation study gave comparable standard deviation values within the test set; thus, we did not include them in the table.

4.5.6 Runtime performance

For each scenario, we compare the runtime performance of the trained models with the reference's, measuring timing for one simulation of 100 frames, averaged over ten different runs. For *ATO*, the computations start with the initial velocity inference by the encoder model and stop when all 100 frames are output by the decoder. All timings were computed using a single *GeForce RTX 2080 Ti* with 11GiB of VRAM.

Table 4.1 shows the summary of computational timings for the reference, baseline (reduced solver without any DNN model), and trained models. For all four cases, our *ATO* model yields improvement in runtime compared to the reference. For the Karman vortex street scenario, our *ATO* model speeds up the computations by 28%, against 29% for *SOL+SR*. Yet, as shown in Sec 4.5.2, *ATO* shows an improvement of the baseline MAE that is 7% better than *SOL+SR*.

	Velocity		Vorticity		Latent space
	MAE	MSE	MAE	MSE	MAE *lerp (ref)*
ATO (ours)	0.133	0.040	0.084	0.015	0.743
no latent loss	0.156	0.057	0.097	0.020	0.488
no encoder	0.259	0.146	0.156	0.48	0.253
no enc. & no lat. loss	0.284	0.174	0.167	0.055	0.322
no solver	0.737	1.073	0.578	0.651	0.713
no adjustment	0.409	0.339	0.212	0.085	0.568
lerp (forces)	0.944	1.725	0.429	0.325	0.682

Table 4.2: Results of the ablation study: we present the MAE and MSE in velocity and vorticity for each model, along with the distance between its reduced space and the down-sampled reference.

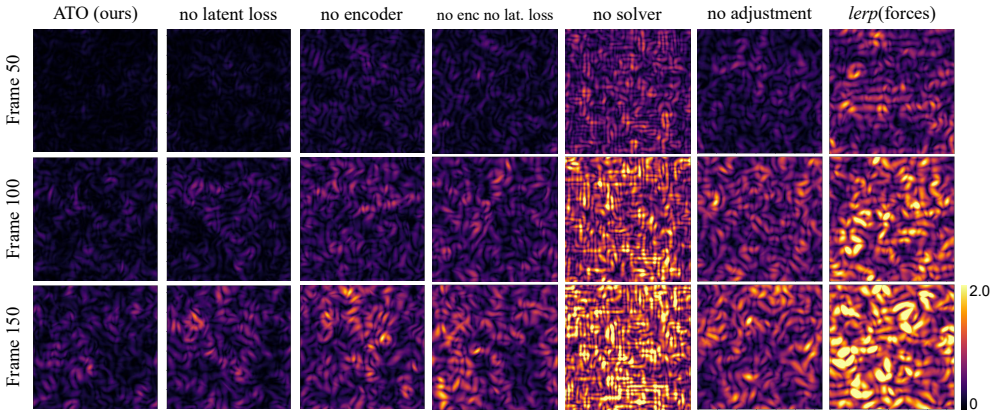

Figure 4.17: Results of the ablation study for one example of the forced turbulence scenario. The absolute error of the velocity field relative to the reference field is shown.

Similarly, for the forced turbulence case, the *ATO* model speeds up the computations by 18%, against 9% for *SOL+SR*, and improves the baseline MAE of 25% more than *SOL+SR*. For the smoke plume scenario, our ATO model speeds up the reference by 20% compared to 28% for the baseline, while improving the baseline MAE by 35%. We note that the *Dil-ResNet + SR* model often has the best runtime performance because it does not contain any numerical solver, but it has errors at least 50% higher than *ATO* and shows very poor temporal extrapolation capabilities.

Training our *ATO* model takes between one and three days depending on the physical scenario, on a *Tesla V100* with 16GiB of VRAM.

4.5.7 Additional visual results

Example sequences of the test data and the inference results of different models for the Karman vortex street scenario are shown in Fig 4.18, for $Re = 850$, along with the spatial distribution of the velocity error in Fig 4.19. Although the visual quality of the different results seems

equivalent at first sight, one can notice that the position of the vortices is more accurate in *ATO*'s outputs than *SOL+SR*'s. Fig 4.20 shows the inference results of the different models for one example of the decaying turbulence case, and Fig 4.21 shows the spatial distribution of the error for the same example. Fig 4.22 and Fig 4.23 show similar results for the forced turbulence scenario.

In our last example, we consider complex flow behaviors created by hot smoke plumes that evolve from random circular densities. We evaluate our model trained with 32 integrated steps on five test simulations with different initial marker fields from which we perform a "warm-up" of 50 time-steps, in order to get interesting plume shapes. Fig 4.24 shows that, despite the increased difficulty of this challenging scenario, our *ATO* model succeeds at reconstructing a complex high-resolution plume of good quality. Indeed, our method presents an improvement of 35% on average over the baseline for 100 steps.

4.6 Limitations and future work

These results show that our training method using the states of physics simulations as latent space of DNNs can facilitate the learning task for complex simulations. This provides a starting point for the exploration of physical latent spaces in many different problems. However, we note that our *ATO* model is not particularly standing out in a simple scenario like the decaying turbulence. Therefore, we can presume that the benefits of its unconventional reduced space are truly visible only when the PDE system is complex enough. In addition to the distance metric, more thorough analysis of latent space contents via, e.g., perceptual metrics, also remains as future work.

Moreover, our method has proven its capabilities in scenarios where force fields were inferred by our networks besides the velocity fields. In the forced turbulence case, the forces are external factors that are independent from the velocity data, thus our *ATO* model has no difficulty finding a latent representation that leads to a superior performance. In Sec. 4.5.7, we showed that our model gives promising results in a scenario where the forces were internal, i.e. created by a marker field that is dependent of the latent velocity. These results are limited however, since our approximation severely diverges from the reference solution after about a hundred time-steps. To tackle this issue, we briefly experimented inferring both the marker and velocity fields together in our networks, but these were not conclusive. That case opens interesting future work, such as finding the best reduced representations for the coupled marker and velocity fields.

Although we evaluated our model on various scenarios, its generalization for broader applications still remains a challenge. For example, our *ATO* architecture does not perform much better than the others on simulations of the Karman vortex street case with Reynolds numbers superior to 1600. We believe that this might come from limitations of the differentiable solver that we used in our experiments, thus training our models with different solvers is also an interesting path for future work.

As our model allows for the production of high-resolution simulations with a reduced solver, it is potentially attractive for editing physics simulations within the learned reduced space in real-time. Indeed, once a coarse initial frame is transformed into *ATO*'s latent space, it is easy to tweak the physical properties of the reduced solver (e.g., viscosity) or to add external factors, such that it can produce high-resolution simulations in a more interactive way. However, the current runtime performance of our model does not allow for such applications. We have not particularly focused on optimizing our code, which leaves room for improvement in terms of performance. But even with such improvements, we did not target very high

resolutions, for the training time was beyond acceptable for resolutions above 256^2. We also made some experiments with 3D data, which led to training times above a month with our current set-ups. Adapting our model for higher resolutions and 3D would thus require a deep revision of our neural networks' architecture.

4.7 Conclusion

In this chapter, we have presented *ATO*, a model that leverages interactions between neural networks and a differentiable physics solver to autonomously explore reduced representations for high-resolution fluid restoration purposes. Our results show that deep neural networks can learn to develop new dynamics for specific learning objectives by using the simulated degrees of freedom as latent space. Our approach opens the path to the exploration of physical latent spaces for other PDEs, as well as different learning tasks than the restoration of details of fluid simulations.

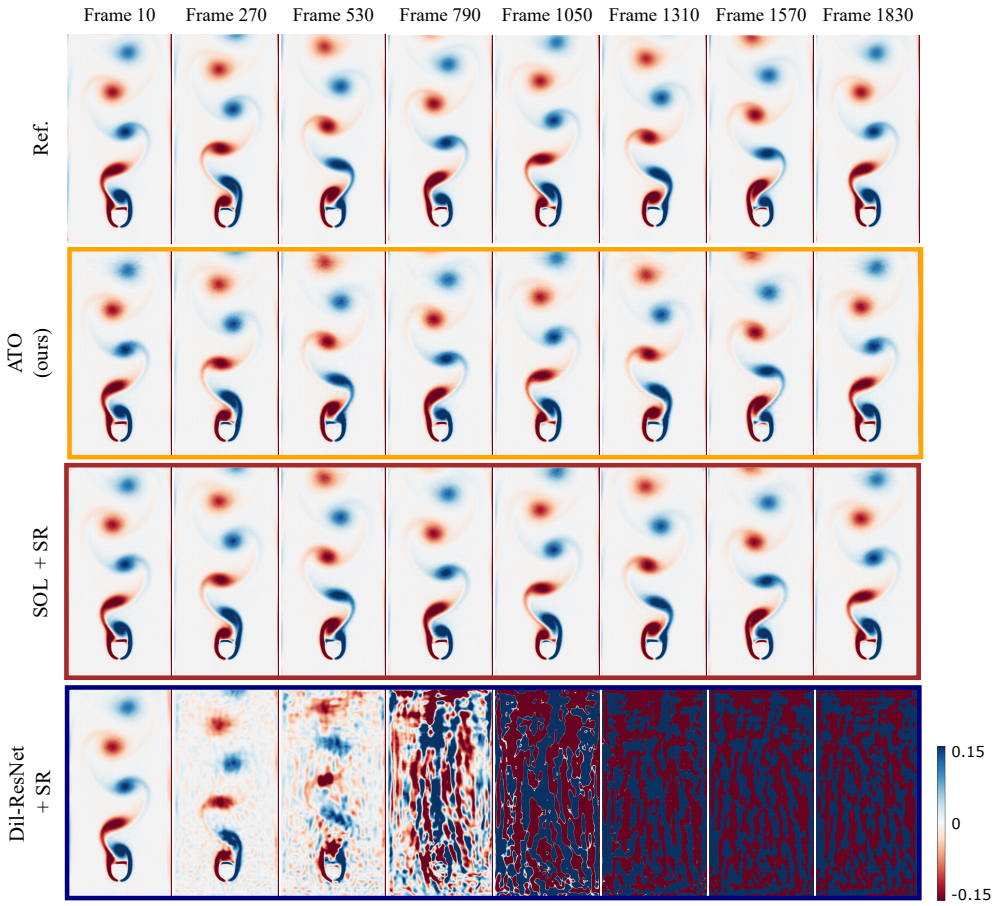

Figure 4.18: Restored frames of different models for the Karman vortex street scenario with Re = 850. The vorticity fields are shown.

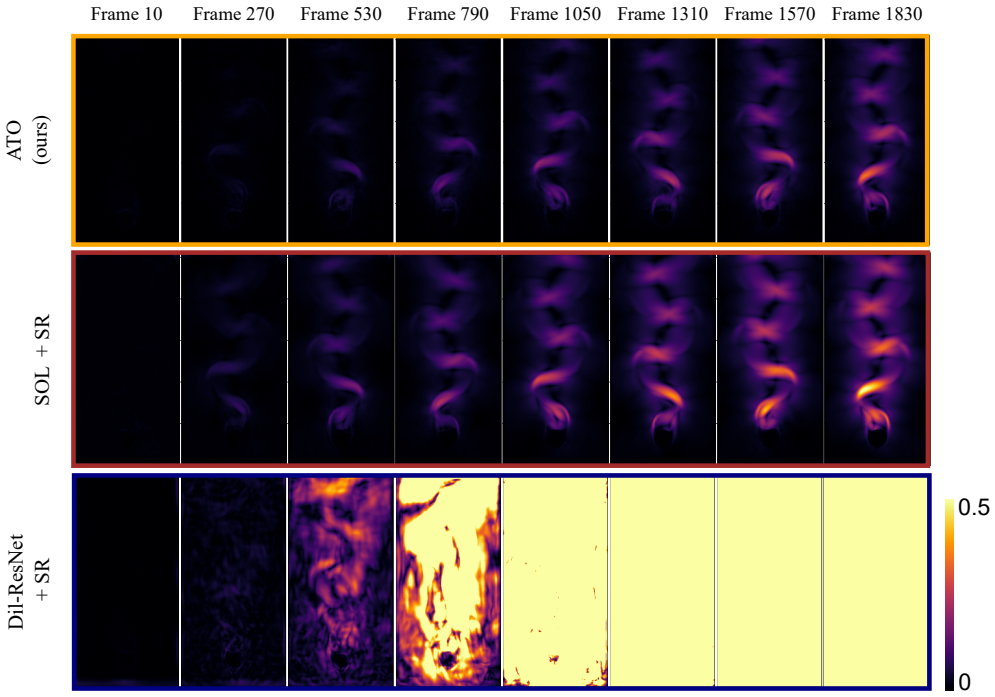

Figure 4.19: Absolute error in velocity for the different models, for the Karman vortex street scenario with Re = 850.

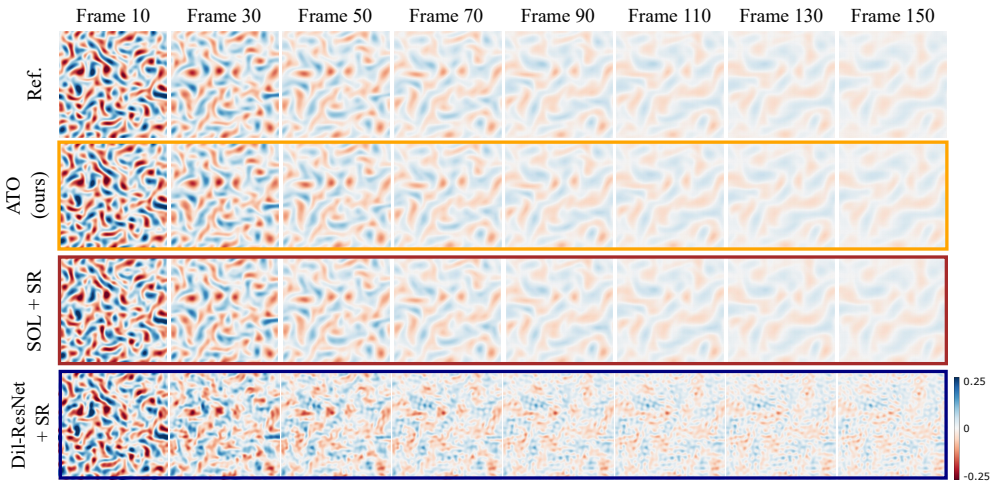

Figure 4.20: Example frames of a test case for different models for the decaying turbulence scenario. The vorticity fields are shown.

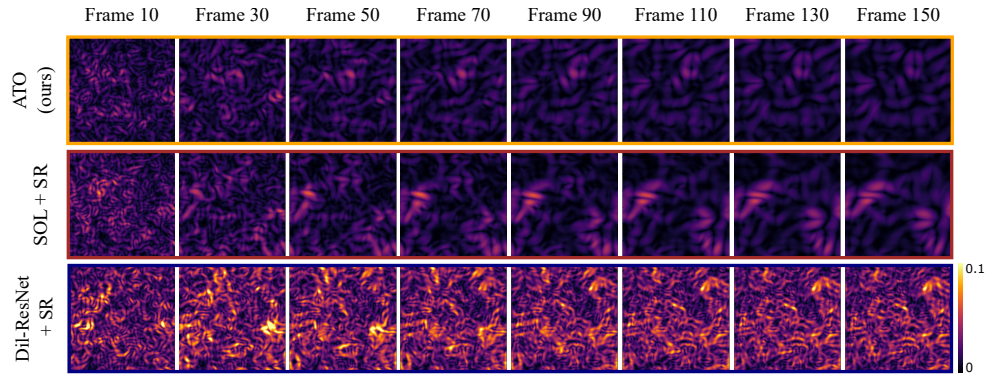

Figure 4.21: Absolute error in velocity for the different models, for the decaying turbulence scenario.

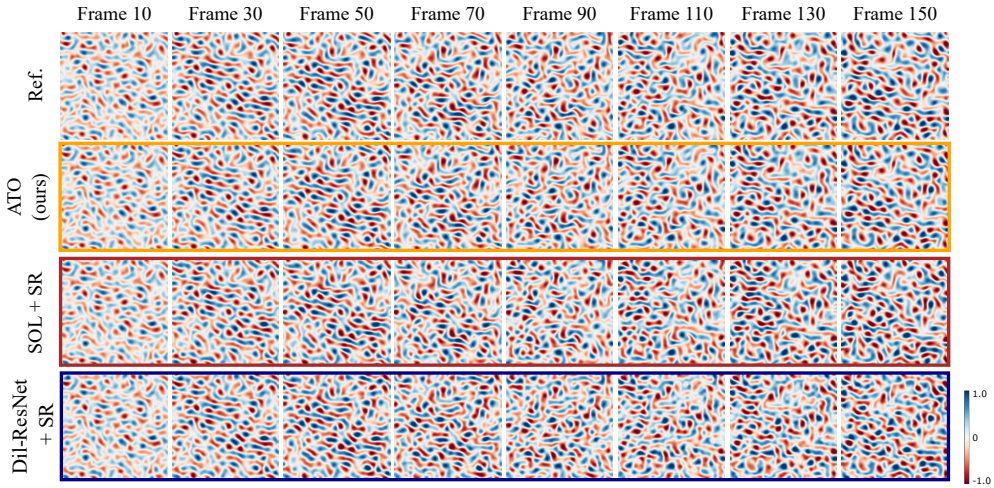

Figure 4.22: Example frames of a test case for different models for the forced turbulence scenario. The vorticity fields are shown.

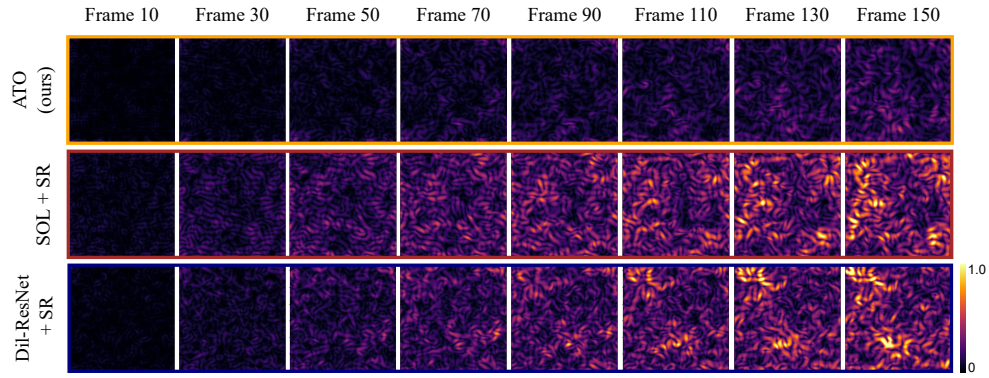

Figure 4.23: Absolute error in velocity for the different models, for the forced turbulence scenario.

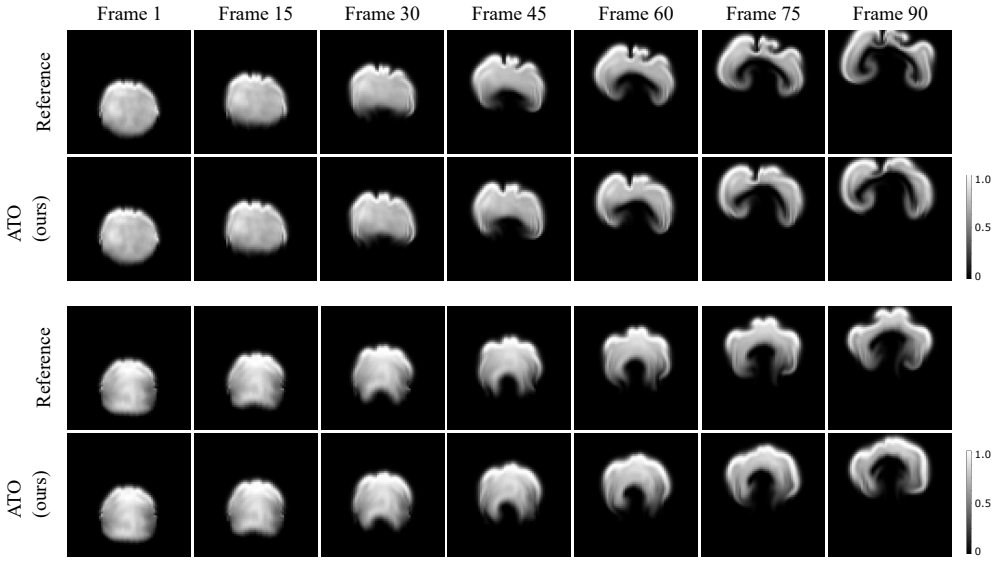

Figure 4.24: Example frames from our *ATO* model for two test cases of the smoke plume scenario. The marker fields are shown.

Chapter 5

Intrinsic SPH simulation on 3D surfaces

We presented our first contribution, which enables the acceleration of Eulerian solvers while showing a better quality performance than previous works. In this Chapter, we will tackle the problem of simulating fluids on surfaces, with a similar goal of using dimension reduction to improve the runtime performance of state-of-the-art SPH techniques on surfaces.

This work is currently under review.

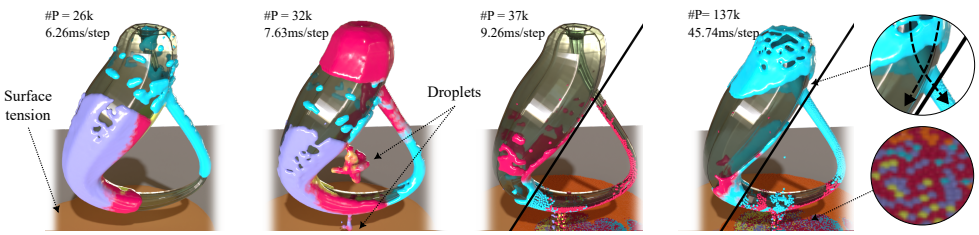

Figure 5.1: We present intrinsic SPH simulation on 3D triangle meshes. Thanks to the versatility offered by standard SPH models, we can simulate surface tension effects as well as droplets formation. Our technique copes with challenging inputs, such as the non-orientable and self-intersecting Klein bottle. Here, particles following trajectories that cross at self-intersections (as shown on the right), do not interfere. Our basic operators allow to simulate scenes with 137k particles at 20 fps, including basic rendering, on a modern laptop (right).

5.1 Introduction

Smoothed-particle hydrodynamics (SPH) have been used successfully for simulating fluids in computer graphics, and in particular in the special effects and animation industries. Simulating fluids and their interaction with deformable objects with high physical fidelity is extremely complex, as it requires solving for the notoriously chaotic Navier-Stokes equations. For that reason, we focus on SPH, which offer a tradeoff between accuracy and efficiency that we believe not to be matched by other simulation frameworks.

The SPH method amounts to representing the fluid using a set of point particles (equipped with physical quantities such as mass, density, pressure and velocity) that interact together through the spatial *splatting* of their carried properties. A basic simulation loop can be summarized as this two-step procedure, repeated over time:

1. For each particle p_i: gather the neighboring particles $\{p_j\}$ in the sphere of influence centered in p_i, and evaluate physical quantities to establish the forces exerted by $\{p_j\}$ onto p_i using the weighted sum derived from the SPH formulae;

2. For each particle p_i: update its velocity and position using a time integration scheme respecting Newton's laws of motion.

These two operations are embarrassingly parallel, and fit perfectly to modern graphics hardware. They merely require simple geometric structures for point-to-point and point-to-object proximity queries (for both influence computation and collision detection), which makes integration into rich code bases simple and easy to maintain.

Physical properties $A(x)$ at point $x \in \mathbb{R}^N$ can be computed by blending the corresponding quantities $\{A_j\}$ equipped on $\{p_j\}$, of mass m_j and density ρ_j, using a smoothing kernel W_h with local support:

$$A(x) := \sum_j \frac{m_j}{\rho_j} A(x_j) W_h(x - x_j). \tag{5.1}$$

Together with the point-representation of the particles, this property makes adding new effects generally straightforward. For example, simulating the interaction between fluids of different viscosity or simulating surface tension amounts to adding corresponding forces accordingly, while simulating *dripping* amounts to simply releasing a tension force onto a particle.

While it is important that the performed simulation remains physically-plausible, explicit control and predictability are often required. In this context, SPH appear as a gold standard, as controlling the fluid amounts to specifying space-time trajectories for points to follow, which is made straightforward by the very point-based nature of the fluid particles. Their simplicity, flexibility, and compatibility with industrial standards have motivated many research works, and many rich and complex simulation effects can be implemented within the SPH framework.

As a disclaimer, our goal is not to make new contributions on fluid simulation, but rather to offer the possibility to *port* existing popular SPH simulation techniques from Euclidean domains to curved domains with intricate topology, allowing for novel rich on-surface simulations. In this chapter, we present a computational framework allowing for robust and efficient SPH simulations on triangle meshes – the de-facto standard surface representation in computer graphics – that copes with common deficiencies in modeled surfaces (non-orientable, self-intersecting, with arbitrary boundaries). We illustrate the effectiveness of our approach by integrating a few common effects in our framework (multi-viscosity simulation, surface tension, droplets) and demonstrate its robustness on challenging inputs (see Figure 5.1), paving the way for flexible intrinsic fluid simulations.

5.1.1 Related work

Fluids in the Euclidean space Fluid simulation in Euclidean spaces (2D and 3D) has been studied for many decades in computer graphics, and many different physics frameworks have been developed to this intent, each coming with ways to balance realism with speed and editability or user control. A detailed review of those different formalisms is done in Chapter 3, and we refer the reader to the Introduction in [88], explaining them in the context of smoothed-particle hydrodynamics (SPH), which have been used extensively in computer animation for their simplicity and flexibility [61, 80]. SPH is a mesh-free Lagrangian particle method, ideal for free surface and interfacial flow problems. It is easy to implement and customize, supporting three-dimensional numerical models naturally. SPH is very GPU-friendly, with real-time solutions for free surfaces and object interactions [101, 92].

Fluids on 3D surfaces Shi and Yu pioneered the simulation of inviscid and incompressible surface flows in [133], adapting previous solvers for the Navier-Stokes (NS) equations from regular 3D grids to arbitrary 3D triangle meshes. In a similar spirit, Fan and colleagues extended the 2D unstructured Lattice-Botzmann method (LBM) to 3D triangle meshes of arbitrary topology [40]. Neill and colleagues then solved the NS equations on dynamic deformable 3D triangle meshes for the first time [105]. Moreover, Auer and colleagues used the Closest Point Method (CPM) to solve the NS equations over triangle surfaces in real-time [7]. However, no specific fluid behaviors are demonstrated and CPM tends to smooth out high frequency details. This approach was later extended to deformable surfaces, although not in real-time [6, 99]. Closer to our use of SPH-based simulations on arbitrary surfaces is the subfield of bubble simulations, including the rich dynamics taking place on their surfaces. Wang and colleagues presented a meshless simulation framework where SPH models the volumes, the surfaces, and surface flows [156]. Deng and colleagues used implicit incompressible SPH from [60] for solving the projection term within their Moving Eulerian-Lagrangian Particle (MELP) framework [34]. Independently from their performance, these works are limited to surface flows that do not consider fluid accumulation nor detachment following the geometry of the surface and properties of the fluid.

Closer to our work, Azencot and colleagues [9] as well as Vantzos and colleagues [154] simulated thin viscous films over 3D triangle meshes using the gradient flow model. Their method worked over static meshes, with a strong dependence on consistent vertex normals. Droplet formation was supported, but not their detachment from and reattachment to the surface. The method did not run in real-time neither. Ren and colleagues derived an integration of the Shallow Water Equation (SWE) with 3D SPH for a stable real-time solution over oriented triangle meshes [117]. Furthermore, Vantzos and colleagues simulated viscous thin films over planar domains, using local-stencils to solve a gradient flow approach [153], and Yang and colleagues solved the NS equation on spherical surfaces [162], both demonstrating real-time performance. Bharadwaj and colleagues proposed the Discrete Droplet Method (DDM) [17], with derivatives approximated using a SPH-like approach, but performance was not real-time. Compared to ours, none of these methods allow for a physically valid interaction between fluid and geometry with real-time user input, over non-orientable meshes.

Intrinsic geometry processing Many works have studied intrinsic operators on 3D surfaces, either to cope with the finite discrete nature of triangle meshes – to go beyond piecewise linear functions, or for robustness reasons – to avoid *remeshing* while benefitting from well-behaved Delaunay triangulations, that avoid badly-shaped elements under point insertion. In the following, we focus on a few concepts that are highly related to the work presented in

this chapter, in particular geodesics and discrete connections for parallel transport on triangle surfaces. More details are given in Chapter 2 and Chapter 3.

Geodesics are the fundamental tool to describe "how far are two points on a curved domain", and are locally-shortest paths connecting points. We are interested in a particular case: the shortest-path connecting two points on a surface (which can be ill-defined: consider connecting opposite poles on a sphere). We refer the reader to [68] for a complete survey. Dijkstra pioneered the search for geodesics by proposing a $N\log(N)$ algorithm to compute shortest paths in graphs [37], and has inspired many subsequent works. While it allowed computing distances restricted to the graph edges, it did not provide geodesics connecting vertices across triangles. However this algorithm has been the basis for many works that proposed exact discrete geodesics, such as the MMP [95] algorithm, which we use in our work. This work analyzes the computation of point-to-point geodesics *crossing edges* only (and *avoiding vertices*), which is the definition we use for our geodesics. As detailed in [68], geodesics are indeed formally well defined across edges only. This is because one can *unfold* trivially adjacent triangles in a common plane, where geodesics then become straight lines, whereas this operation becomes ill-defined when crossing vertices (as there are infinite ways to keep walking inside neighboring triangles after leaving the vertex). Incidentally, even in works based on Discrete Exterior Calculus (DEC) or Finite Elements Methods (FEM), this corresponds to the notion of discrete connection that is mostly used [116, 31] in order to parallel transport vectors across adjacent triangles. In order to define a consistent geometry processing framework for gathering and averaging transported tangential quantities, we borrow those definitions of discrete geodesics and trivial connections, which result in consistent discretizations even on ill-defined surfaces. As a result, we are able to implicitly unfold non-orientable, and/or self-intersecting triangle meshes with arbitrary boundaries, and compute logarithmic maps on them (which is prevented by previous methods requiring global solves on triangle meshes [131, 141]).

5.1.2 Contributions

We introduce a simple and sound computational framework, consisting of two canonical operators:

- Geodesic neighborhood's gathering and averaging of scalar/tangential quantities, see Section 5.2.2;
- Intrinsic walk on a 3D mesh (see Section 5.2.4).

We specialize these operators for the case of intrinsic SPH simulation robust to pathological inputs, that require *fast and parallel* computations on *small* neighborhoods. Our framework offers the following advantages:

- it is optimization-free and scalable;
- it complies with low-quality triangle meshes;
- it makes little assumptions on the connectivity: we merely require that no more than two triangles share a given edge.

We demonstrate the robustness of our technique on complex examples, featuring variable geometric detail as well as self-intersecting and/or non-orientable structures with arbitrary boundaries (e.g. the Moebius strip or the Klein bottle).

5.2 Method

Our technique takes as input a triangle mesh $\mathscr{S}$, with the only connectivity constraint that each edge should be connected to at most two triangles, as our intrinsic operators require to *walk* from triangle to triangle on the mesh across shared edges only. In Chapter 2, we presented the formalism for the restricted class of geodesics that we consider, and we first add a few necessary operators in Section 5.2.1. We then introduce our *neighborhood gathering and averaging* operators in Section 5.2.2, and finally our *walking* operator in Section 5.2.4. We present our application to SPH simulation on 3D surfaces in Section 5.3.

5.2.1 Mathematical notations

In this section, we do not extensively remind the connectivity structure that we equip our triangle meshes with, but complete the notions presented in Chapter 2. To comply with a large variety of input surfaces, we consider triangle-triangle adjacency through shared edges only, and merely require that no more than two triangles share a given edge. Our triangle structure, detailed in Listing 5.1, requires storing little information of fixed size, which makes GPU storage and traversal straightforward.

Listing 5.1: Our triangle structure.

```
struct Triangle {
    int corners[3]; // Vertex indices
    int neighbors[3]; // -1 if boundary edge
    int neighbors_flip[3]; // Are ngbring tri. oriented consistently
    mat2x2 T[3];
    mat2x3 P;
};
```

Given two triangles t_1 and t_2 (an example triangle is shown in Figure 5.2), we define a rotation matrix $\mathscr{T}_{t_1t_2}$ mapping tangent vectors in t_1 to tangent vectors in t_2 in the 3D space. One can thus transport a vector from t_1 to t_2 in (δ_u, δ_v) coordinates using the following 2D map

$$T_{t_1t_2} := P_{t_2} \mathscr{T}_{t_1t_2} E_{t_1} \in \mathbb{R}^{2\times 2} \tag{5.2}$$

with

$$\begin{aligned} E_t &:= (\boldsymbol{e}_u | \boldsymbol{e}_v) \\ P_t &:= (E_t^T E_t)^{-1} E_t^T \end{aligned}$$

This map is stored as `T` in Listing 5.1.

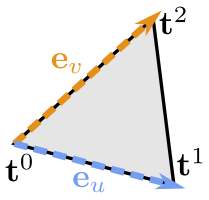

Figure 5.2: Triangle $t = (\boldsymbol{t}^0, \boldsymbol{t}^1, \boldsymbol{t}^2)$, with $\boldsymbol{e}_u =: \boldsymbol{t}^1 - \boldsymbol{t}^0$ and $\boldsymbol{e}_v =: \boldsymbol{t}^2 - \boldsymbol{t}^0$

Boundary reflections Boundary conditions for fluids can be set in various manners. A common practice in SPH simulations is to create *virtual particles* along boundaries (or colliding objects) to repulse the fluid's particles. In our case, we implement a *bouncing* boundary condition, by transforming an incident tangent vector $\boldsymbol{\delta}_t$ as

$$\boldsymbol{\delta}'_t = \left(I - 2\frac{(\boldsymbol{n}_t \times \boldsymbol{e})(\boldsymbol{n}_t \times \boldsymbol{e})^T}{\|\boldsymbol{e}\|^2} \right) \boldsymbol{\delta}_t =: \mathscr{B}^t_{\boldsymbol{e}} \boldsymbol{\delta}_t \tag{5.3}$$

for particles bouncing on a boundary edge $\boldsymbol{e}$, with $\boldsymbol{n}_t$ the 3D normal of the triangle. Like for discrete connections, we can express this as a 2D map transforming a tangent vector's (δ_u, δ_v) coordinates in t using

$$B^t_{\boldsymbol{e}} = P_t \mathscr{B}^t_{\boldsymbol{e}} E_t \in \mathbb{R}^{2\times 2} \tag{5.4}$$

For compactness, we store this information in `T` as well in Listing 5.1, for boundary edges.

5.2.2 Neighborhoods computations

Using the computational framework just introduced, we present our construction of geodesic neighborhoods, that we target for on-demand computation of geodesic length and combined geodesic parallel transport within a specific geodesic radius h (given by the local influence radius of the SPH kernel, see [101]).

Our method is based on the MMP algorithm [95], that uses the so-called *window propagation mechanism*. This algorithm is highly compatible with our setup in practice, as we consider discrete geodesics along shared edges only, which is the way windows are propagated in the MMP algorithm.

In a preprocess, we compute the geodesic windows from the center of each triangle, at a conservative distance: considering a triangle t with center $\boldsymbol{c}$ and circumcenter r_t (distance from $\boldsymbol{c}$ to t's corners), and given a SPH kernel radius h, we compute geodesics from $\boldsymbol{c}$ inside the geodesic disk of radius $h + r_t$. We reviewed the basics of the MMP algorithm in Chapter 2, and now present our extensions.

MMP extension We extend the MMP algorithm in several ways:

1. While propagating/splitting windows across an edge e shared by adjacent triangles t_1 and t_2 (conceptually marching from t_1 inside t_2), we record the transformation $T_{t_1 t_2}$ (Eq. (5.2)), and concatenate it with the series of rigid transformations from the source triangle t, in order to align t_2 with respect to t. This captures the direction $\vec{d}_{\boldsymbol{cq}}$ from the center of t, $\boldsymbol{c}$, to a point $\boldsymbol{q}$, i.e., the normalized tangent of the geodesic at $\boldsymbol{c}$.

2. During propagation, the first time a saddle point is encountered, we keep the direction of the geodesic from the source to the point to serve as direction of the geodesics at the source.

Geodesic distances can therefore be computed easily from the position of a point and the associated window information. For our particle simulation we also need geodesic directions that can be computed either using accumulated rigid transformations or by fetching the direction if a saddle point was encountered. This also makes the application of parallel transport of vectors straightforward, as needed in Section 5.2.3, since we simply need to apply the accumulated transformation matrices along the geodesic path between the vectors.

During this pre-computation step, we register for each triangle the geodesic windows starting from its center, in a radius $h + r_t$. This enables us to get the neighborhood of each

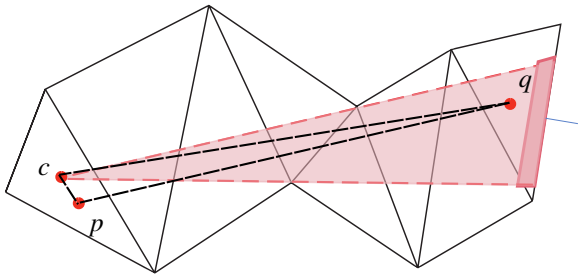

Figure 5.3: To get the distance between arbitrary points $\boldsymbol{p}$ and $\boldsymbol{q}$ from different triangles, we use the geodesic window from the center of $\boldsymbol{p}$'s triangle, $\boldsymbol{c}$, to $\boldsymbol{q}$. We *triangulate* the location of $\boldsymbol{p}$ with respect to $\boldsymbol{q}$ realigned in $\boldsymbol{p}$'s triangle.

triangle, i.e. the triangles that these windows are going through. We store the corresponding data in buffers on the GPU (the neighborhoods of all triangles being of variable size, we build a compact indexed structure to access it).

Our structure allows capturing geodesics from *the center of a triangle* t to any point on $\mathscr{S}$ in constant time. However, we cannot capture directly the geodesics from *arbitrary points in* t to arbitrary points on $\mathscr{S}$. We make an approximation to compute those, using the quantities we derived. Given an arbitrary source point $\boldsymbol{p} \in t$ and a target point $\boldsymbol{q}$ inside a given window (see Figure 5.3), we consider $\boldsymbol{p}$, $\boldsymbol{c}$ (the center of t), $g_{\boldsymbol{cq}}$ (the geodesic window from $\boldsymbol{c}$ to $\boldsymbol{q}$) and $\vec{d}_{\boldsymbol{cq}}$ (the geodesic direction) to *triangulate* the location of $\boldsymbol{p}$ with respect to the point $\boldsymbol{q}$ realigned inside t. Doing so, we make the assumption that the discrete geodesic from $\boldsymbol{p}$ to $\boldsymbol{q}$ passes through the window corresponding to the geodesic from $\boldsymbol{c}$ to $\boldsymbol{q}$.

This obviously results in approximate geodesics and transported directions only, but we note that this approximation becomes accurate for small distances. The SPH kernel decreasing with the distance, the approximation that is made is in practice harmless for our applications. We analyze this claim in Section 5.4.

Thanks to the previously defined neighborhood of each triangle, of radius h, we are able to find the neighbors of every particle by querying the geodesic distance to the particles lying in these neighboring triangles.

5.2.3 Velocity and forces update

In the previous section, we described how the neighborhood of each particle can be found. Thanks to our geodesic distance computation, we can then evaluate the necessary physical quantities and proceed to the next steps of the SPH algorithm (see Chapter 2 for more details).

The density and pressure computations are straightforward, since the only difference with traditional SPH is that geodesic distances are queried in the kernel function instead of Euclidean ones. These physical properties are then used to compute the internal and external forces applied to the particles. These forces rely not only on the distance between particles, but also on the direction from one to the other. For instance, when we calculate the influence of particle p_j on particle p_i, we need to transport the velocity v_j from p_j to p_i along the surface. This is once again straightforward in our case since the algorithm described in the previous section provides us with the geodesic path and accumulated transformation matrices that we need to parallelly transport such vectors.

Therefore, we are able to correctly accumulate the influence of all neighboring particles of p_i by making all our computations in its tangential plane. Finally, we project external forces

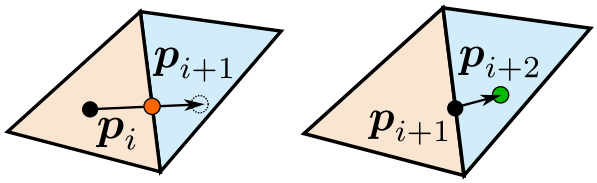

Figure 5.4: If a particle needs to leave its current triangle, its position is snapped on the closest positive intersection of its trajectory with the triangle edges. The velocity is transformed in the adjacent triangle and the walk continues.

that are not in the triangle plane, like gravity, thanks to Equation 2.3.1, to get the acceleration of the particle.

That way, we can apply Newton's second law of motion and update p_i's velocity field. To do so, we transport the velocity at the previous time-step in p_i's current triangle, since it might have moved from one triangle to another. We can do so by registering the necessary transformations from one triangle to the other during the walk of the particle at the previous time-step. Then, we can update the velocity with the acceleration since they lie in the same plane. We use a *leap-frog* scheme, thus this new velocity is used for the *walk* algorithm described in the following section.

5.2.4 Walk

In this section, we detail our on-surface *walk* algorithm. We let a particle have two possible states: it can be strictly inside a triangle or on an edge. Since we define the connectivity of our surface by edge adjacency, we never let a particle be on a triangle corner. Before starting its walk on the surface, a particle carries multiple information: a triangle index *triIdx*, a topological flag (e_i, e_j) indicating which edge it lies on ($(-1,-1)$ if strictly inside a triangle), barycentric coordinates (u, v), and a 2D velocity (δ_u, δ_v). Our *walk* algorithm (detailed in Algo 2) starts by an update of the barycentric coordinates applying Newton's laws of motion. Then, four situations can arise:

1. If the particle goes in a new triangle, we compute the intersection of its trajectory with the triangle edges and snap the particle's position on the closest positive one (see Figure 5.4), denoted as λ.

2. If the intersection λ is inside a *corner zone*, we snap the particle's position on the intersected edge at a distance ε of the corner, on the border of the corner zone, as shown in Figure 5.5.

3. If the particle's updated position stays inside the same triangle as at the previous step, the walk is finished. We update the particle's barycentric coordinates and perform the next step of SPH.

4. If the particle's updated position stays inside the same triangle, but goes in the *corner zone* of this triangle, we proceed as in (2). We then update the particle's barycentric coordinates and perform the next step of the SPH.

In cases (1) and (2), we update the topological flag of $\boldsymbol{p}_i$, indicating that the particle now lies on an edge, we update its *triIdx* to the adjacent triangle index, and finally transform the velocity (δ_u, δ_v) with the corresponding T matrix. If the particle did not walk its full distance,

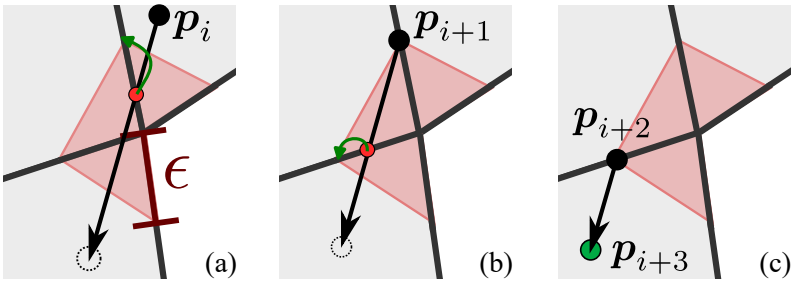

Figure 5.5: (a) If the particle trajectory crosses a corner zone (red), p_i is snapped at a distance ε of the corner and the walk continues (b) until it goes out of the *corner zone* (c).

the walk continues with $\Delta t = \Delta t - \lambda$. When $\Delta t = 0$, the walk ends and the next SPH step can be performed.

With this algorithm, an intersection is always found when a particle needs to leave its current triangle, and a particle never gets stuck on a triangle vertex. Therefore, we ensure that our *walking* algorithm always finishes.

5.3 Results

5.3.1 Implementation details

We implemented our intrinsic operators on the GPU in compute shaders, leading to an interactive SPH simulation with hundreds of thousands of particles (see Table 5.1 for statistics). We pre-compute the connectivity information of the mesh on the CPU only once at loading time, along with the particles' initial state and a pre-filtering of their triangle's neighborhood. Indeed, we pre-compute the neighborhood of each triangle by iteratively going over its adjacent triangles and adding the ones inside the SPH kernel. Then, the particles being constrained to the surface, we only query the ones that are in neighboring triangles during neighborhood computation on GPU. To optimize memory usage, the particles are ordered on GPU according to their *triangle index*.

Once the neighborhood of each particle is set, we perform the traditional SPH stages of computing density, pressure, and forces. To do so, we re-normalize the typical kernels used in 3D, since we focus on surfacic (*i.e. 2D*) simulation. Then, we update the particles' velocities and positions using an *explicit Euler* scheme. The forces applied to one particle being calculated in its tangential plane, the updated velocity can directly be used for the *walk* of the particle (Sec 5.2.4). This last step only requires the particle's triangle index, topological flag, and barycentric coordinates, along with the connectivity information of the mesh, which are all already available in memory.

5.3.2 Intrinsic SPH simulation

Our intrinsic operators leverage all the benefits of Lagrangian methods, and are compatible with various effects of SPH simulations (Figure 5.10, 5.11, 5.12, 5.13). In this section we illustrate a few of them, including droplets (Figure 4.1), surface tension (Figure 5.11, left), and interactions between different types of flows (Figure 5.11, right). Our droplets simulation relies on a *density threshold* above which a particle can leave the surface to free-fall in the 3D space. Since the SPH stages are not performed for these particles, they are attached to an

ALGORITHM 2: Step i of our on-surface walk operator

Input: particle: triIdx, edge, $\{u_i, v_i\}$, $\{v_{u,i}, v_{v,i}\}$.
while *WalkFinished = FALSE* **do**
  $\{\tilde{u}_i, \tilde{v}_i\} = \{u_{i-1}, v_{i-1}\} + \Delta t_{i-1} \times \{\delta_{u,i-1}, \delta_{v,i-1}\}$;
  if $\tilde{\alpha}_0 \in (0, 1-\varepsilon)$ *and* $\tilde{\alpha}_1 \in (0, 1-\varepsilon)$ *and* $\tilde{\alpha}_2 \in (0, 1-\varepsilon)$ **then**
    edge = {-1, -1 };
    $\{u_i, v_i\} = \{\tilde{u}_i, \tilde{v}_i\}$;
    WalkFinished = TRUE;
  else
    Intersect(triEdges, $\{u_{i-1}, v_{i-1}\}$, $\{\delta_{u,i-1}, \delta_{v,i-1}\}$);
    Keep closest positive intersection λ;
    triIdx = adjacent triangle;
    edge = intersectedEdge;
    $\{u_i, v_i\} = \{u_{i-1}, v_{i-1}\} + \lambda \times \{\delta_{u,i-1}, \delta_{v,i-1}\}$;
    for *k in range 3* **do**
      if $\alpha_k > 1-\varepsilon$ **then**
        $\alpha_k = 1-\varepsilon$;
        break;
      end
    end
    Compute $\{u_i, v_i\}$ in new triangle;
    $\{\delta_{u,i}, \delta_{v,i}\} = T \times \{\delta_{u,i-1}, \delta_{v,i-1}\}$;
    $\Delta t_i = \Delta t_{i-1} - \lambda$;
    WalkFinished = FALSE;
    if $(\Delta t_i = 0)$ **then**
      WalkFinished = TRUE;
    end
  end
end

additional ghost triangle, in order to optimize compute time in the shaders. We added some randomization in the dropping test, so that the fluid can reach a new equilibrium state on the surface without losing all its particles. Finally, if a new intersection with the surface is found during a particle's free fall, it is attached back to the mesh and our operators are applied again on that particle.

Our framework is compatible with a large variety of meshes, as long as an edge is not shared by more than two triangles. Figure 5.12, right, illustrates our results on a Moebius strip, where particles flow all along the strip and bounce back at the borders. Figure 5.12, left, illustrates how particles do not to interfere at self-intersections on the Klein bottle and instead follow their own trajectory along the mesh. Indeed, our neighborhood computations with geodesic distances do not consider such particles to be neighbors, whereas a typical Euclidean framework would. Figure 5.9, greatly illustrates the benefits of our method upon Euclidean neighborhoods, particles slowly flowing along the paper folds in the former, instead of getting stuck as in the latter.

5.4 Analysis

5.4.1 Memory usage and performance

Memory consumption In order for the particles to have access to their neighbors, our implementation requires that the underlying triangles of the mesh store their neighbors (adjacent triangles), which has an impact on the memory consumption depending on the overall topology of the mesh (see Table 5.1). For instance, the *Folded paper* scene contains hundreds of large elongated triangles that are connected to a large set of other triangles, which explains the high memory consumption. For particles, our framework requires very few additional data when compared to traditional SPH simulations: the triangle index to which the particle belongs, the edge topological flags, and the barycentric coordinates, which leads to three integers and two floating point values. We note that we did not spend time on optimizing the memory layout of our data structures, which currently have redundancies (especially regarding neighboring triangles) that could be further optimized.

Performance Our pre-processing step (Section 5.3.2) can take up to a few minutes and is done only once at startup. It is currently performed on a single thread on the CPU, and could easily be implemented in a multi-threaded context for more efficiency. After that, our simulation runs interactively on the GPU for scenes featuring up to thousands of particles.

As with classical SPH simulations, the performance is mainly dependent on computations that involve neighboring particles, thus it is related to the SPH kernel radius h. In our case, a triangle neighborhood is defined as a geodesic disk of radius $h + r_t$, with r_t the circumcenter of the triangle. This makes both the pre-processing step and the particles neighborhood computations highly dependent on the kernel radius and overall topology of the mesh. Having a mesh with isotropic triangles is thus a crucial factor for the simulation efficiency. This explains the timings for the *Folded paper* scene, which features many thin and elongated triangles especially around the folds. On the other hand, the *Table* scene mostly contains isotropic triangles that greatly favor performance for similar amounts of triangles and particles.

Scene	#T	#P	E	Memory (Gb)		Preprocessing		Runtime		
				Mesh	Particles	Kernel radius	Time (s)	Av. step (ms)	#Steps	Total (s)
Moebius strip (Figure 5.12)	448	2323	0.37	0.002	0.005	0.15	0.02	3.9	1000	4
Klein bottle (Figure 5.12)	9232	38 488	0.58	0.03	0.08	0.05	1.1	16	1131	18.5
Spring (Figure 5.13)	43 786	19 544	0.09	1.1	0.04	0.05	33.4	6.8	5000	34
Folded paper (Figure 5.9)	82 856	128 707	0.10	2.6	0.28	0.05	154.4	90.2	2000	171.8
Table (Figure 5.13)	115 432	94 970	0.12	0.43	0.21	0.05	313.6	33.46	3000	100.4

Table 5.1: Performance statistics for the different scenes shown in the paper: triangle count #T, particle count #P, average edge length in the mesh E, GPU memory consumption of both the mesh and the SPH particles, SPH kernel radius, preprocessing time (s), average time for a single step (ms), total number of steps, and total time for the entire simulation (s).

5.4.2 Approximations

In this section, we discuss the approximations made in our work and sketch possible avenues for future work.

Kernel normalization The first approximation we make is about the renormalization of the kernels we use for the SPH simulation. By setting the normalization factor for kernel W_h as

$$\|W_h\| := \int_{r\in\mathbb{R}^2} W_h(r)dr,$$

we assume here that the underlying surface geometry around a point x is *flat*, and given by a disk at least larger than h. Formally, one could normalize the kernel at point x using

$$\|W_h\|(x) := \int_{y\in\mathscr{S}} W_h(y-x)dy$$

instead, i.e., capturing the surface geometry, and integrating W_h on the geodesic disk centered in $x \in \mathscr{S}$. Note however that it would lead to *asymmetric* influence weights

$$\frac{W_{ij}}{\|W_h\|(p_i)} \neq \frac{W_{ji}}{\|W_h\|(p_j)},$$

breaking an important assumption of SPH simulations, which is *reciprocity* (i.e., particles p_i and p_j influence each other symmetrically).

Shortest-path influences Another approximation made in our work is that particles see each other along the shortest-path geodesic connecting them. Equivalently, each particle sees its neighborhood through a local logarithmic map parameterization. While this is a rather natural idea, one can observe that shortest-path geodesics may be unstable in thin regions, for example along tubular regions where the local feature size (distance to the medial axis) is small with respect to the kernel radius h. In these situations, particles may bounce in an unstable manner, as the shortest-path geodesic connecting them may keep jumping iteration after iteration.

Geodesic approximation As explained in Section 5.2.2, we only compute exact geodesics (up to numerical imprecisions) from the centers of the triangles to a sub-part of the mesh surface – as we only request geodesic neighborhoods in the support radius h of the kernel. To establish an estimate of the distance to an arbitrary source point in a triangle, we rely on a simple approximation. In Figure 5.6, we illustrate the errors introduced on the resulting geodesic distance. As illustrated, the errors are mostly located far away from the source. In our application scenario, these errors are negligible as those correspond to points outside the support of the SPH kernel. Figure 5.7 illustrates our approximate geodesics and parallel transport on the Moebius strip as well as on a noisy Klein bottle. We also show in Figure 5.8 a logarithmic map computed on the Moebius strip. As we do not enforce smoothness, we observe discontinuities in the argument, revealing the presence of saddle points on the mesh.

Neigborhood structure size Our framework highly relies on GPU computations in order to run at interactive speed, which also has its set of drawbacks. The main issue that we face is the diffculty to allocate memory dynamically. Indeed, SPH requires that we recompute every particle's neighborhood at each simulation step, which thus has a variable size. This is not compatible with the pre-processing step in which we allocate memory for the neighborhood

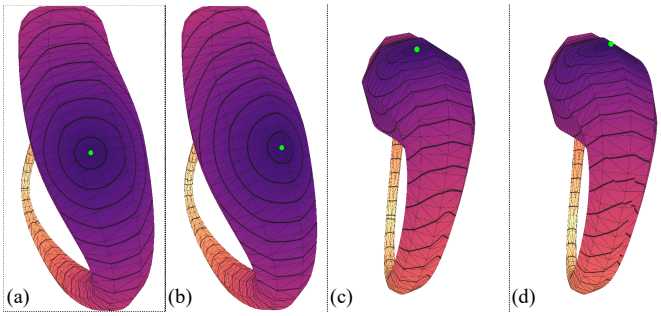

Figure 5.6: We show our geodesic distances to a point highlighted in green on the Klein bottle. The distances to the center of the source triangle are shown in a) and c), and our approximation of the distance to a point far from the center are shown in b) and d). One can notice discontinuities resulting from our approximation, which are in practice acceptable in our case since we focus on local neighborhoods.

arrays only once, when transferring the data from CPU to GPU. These arrays have a fixed size, which can either be way too big for particles with low density, or too small in the opposite case. In the first case, the only issue is about performance, which we tackle by ordering the particles by triangle index. This way, particles with low density should be processed together in the compute shaders and do not need to "wait" for each other. In the other case, where particles have more neighbors than is allowed by the array size, physical errors can arise. Indeed, randomization is introduced in the neighborhood choice in this scenario, which can lead to incorrect behaviors, and might create a snowball effect with particles nearby. A more thorough analysis of memory usage and neighborhood size could be conducted in order to fix these potential issues.

Other approximations In Section 5.2.4, we present a *corner zone* to prevent numerical instabilities that can occur on triangle vertices when using intrinsic frameworks. This zone is delimited by segments of size ε on the triangle edges (Figure 5.5), currently set to 10^{-4} in barycentric space. No particle can enter this zone, which results in physical approximations that are barely visible when a particle slides on the surface. It can however create some instabilities when the flow reaches equilibrium locally on the surface, for example if a big amount of fluid is accumulating on a small area or on a border.

Finally, if a particle p_i is located on an edge e and its velocity (δ_u, δ_v) is colinear to e, we slightly shift the velocity inside the triangle. This error can propagate if many steps are performed for the walk during one simulation time-step, which is not our case as showed in Section 5.2.4.

5.5 Conclusion

We have presented a simple and robust computational framework to perform intrinsic SPH simulations on 3D surfaces, that allows considering challenging inputs such as self-intersecting and non-orientable surfaces with arbitrary boundaries. We have demonstrated that a typical SPH implementation with our method can simulate various effects at interactive speed, such as surface tension, multi-viscosity simulations and droplets.

Although our intrinsic simulations behave empirically as their Euclidean counterparts (in flat spaces), the impact of simulating SPH on curved domains using a symmetric point-wise

method remains to be analyzed, which echoes our discussion on kernel renormalization. While it results in the non-uniform treatment of particles, we did not observe visible artifacts that could be linked to this in practice, even in highly-curved geometries (see the Spring example). This may be due to the built-in smooth nature of SPH. Moreover, we use the input triangulation for nearest neighbor queries, which we treat in spirit as a hash map. We note that we have not investigated in depth the relationship between the physical properties of our fluids (leading to target inter-particle distance) and the size of the triangles which we use for our simulation, which clearly impacts performance. While it is always feasible to subdivide the input triangles if too many particles are located on one of them (refining the neighborhood query structure without changing the geometry of the surface), merging many small triangles together in the opposite situation will change the surface geometry and could result in incorrect simulations. Designing a proper intrinsic multi-resolution structure adapted to our problem will be key to further improve performance and scalability of our approach.

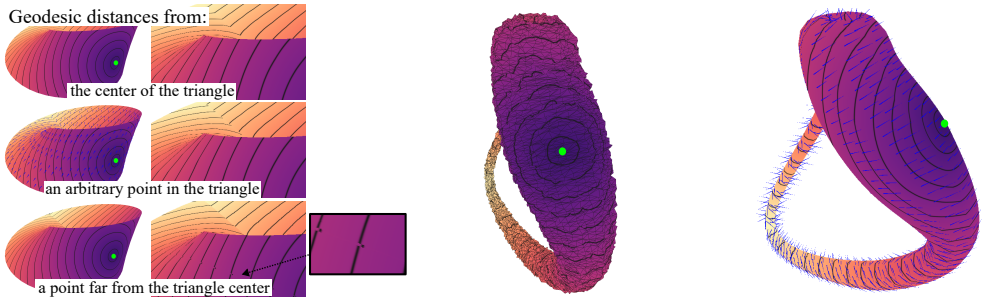

Figure 5.7: Our algorithm allows computing geodesics as well as associated parallel transport of tangent vectors on triangle meshes, as long as no more than two triangles share an edge, regardless of non-orientability.

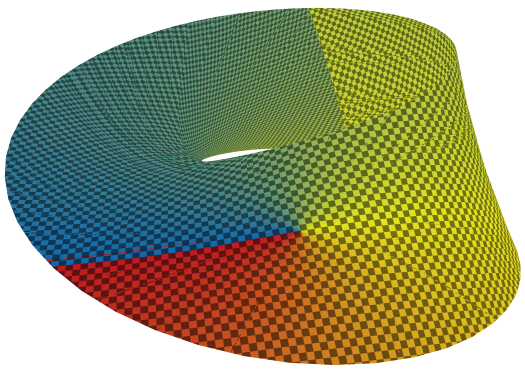

Figure 5.8: Our algorithm allows computing approximate logarithmic maps on surfaces.

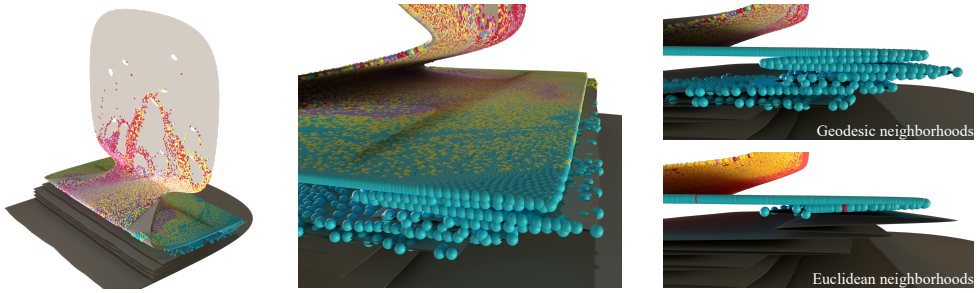

Figure 5.9: Considering Euclidean neighborhoods and distances (bottom right) prevents computing robust intrinsic simulations, as soon as Euclidean and geodesic distances differ too much, even in the absence of self-intersections (for which particles close in the Euclidean space wrongly interact and prevent the flow from falling along the folded geometry). Our intrinsic simulation behaves exactly as if this developable surface was unwrapped in the plane, as expected.

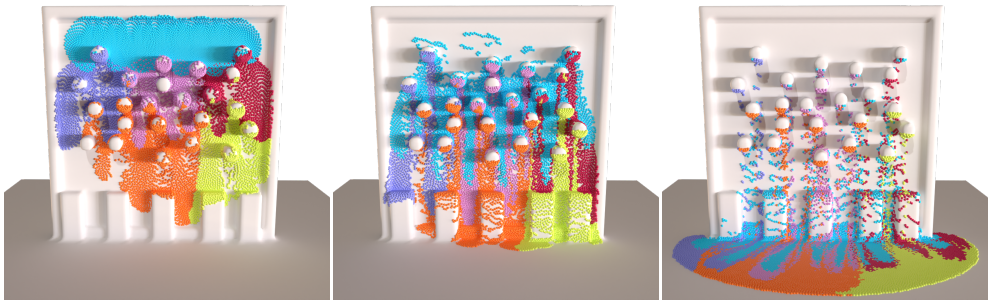

Figure 5.10: Three frames of a simulation with 100k particles dripping down the vertical part of the mesh, slowly flowing on the horizontal plane.

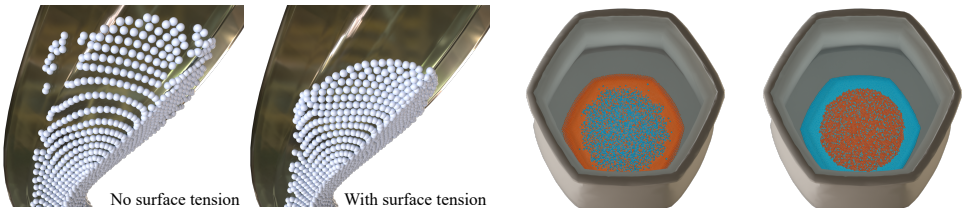

Figure 5.11: Our approach is compatible with a standard SPH method with various effects. Left: comparisons with and without applying surface tension. Right: dropping a low-viscosity fluid (blue) on top of a high-viscosity one (orange) is shown on the left part, and the opposite experience is shown on the right. Here, the orange particles remain glued together, and progressively force the blue particles to move to the side of the vase.

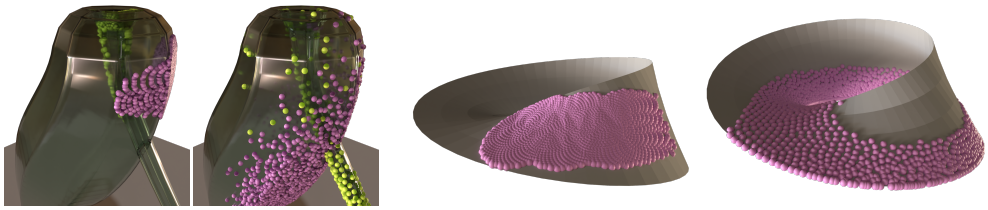

Figure 5.12: Thanks to the intrinsic nature of our method, the simulation can be performed on non-orientable and/or self-intersecting surfaces. Left: the green and purple particles do not interact on the Klein bottle. Right: the liquid flows on the Moebius strip without showing any discontinuities and correctly handling boundaries.

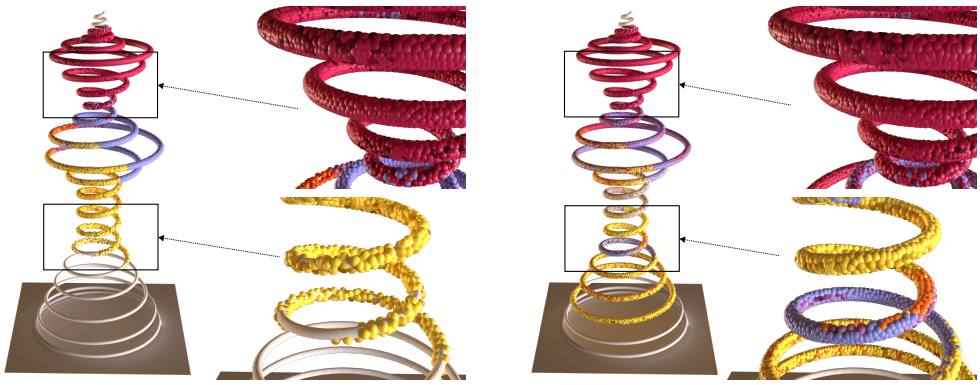

Figure 5.13: Our method handles very thin structures such as strand-like meshes. In this example, 43k particles are slowly following a standing spring-shaped object and moving down towards the ground.

Chapter 6

Conclusion

In this thesis, we introduced several contributions to the field of computer graphics, and more specifically fluid simulation. We aimed at reducing the computational costs and resources required to simulate such natural phenomena in a realistic way, and presented two novel techniques that were using dimension reduction as a means to do so. In this chapter, we will first summarize our contributions and then present the research perspectives that they open, as well as more general societal perspectives concerning computer graphics research.

6.1 Exploring physical latent spaces

6.1.1 Contributions

Firstly, in Chapter 4, we presented a data-driven technique that enabled the production of turbulent fluid simulations from a unique coarse frame, at a higher-resolution than given. We proposed a deep learning model that enabled the exploration of reduced latent spaces, without having any other constraint than fitting a target solution. Our model, named *ATO*, is composed of three networks: an encoder, an adjustment and a decoder model. It was trained on and applied to turbulent flows such as the Karman vortex street scenario, a smoke plume or a turbulent scenario with external forces. We compared our results to two previous works that both operate in a low-resolution space, either using a deep neural network to correct the numerical errors created by a coarse solver, or by training a surrogate model that replaces the numerical solver completely. We showed that our model improved the performance of both existing works in several physical scenarios, by finding a latent space that is optimized for applying the reduced solver and decoding the frames at a higher resolution. Taking a closer look at that latent space, we found that it was quantitatively very far from the bilinear down-sampling of the ground truth. However, it was easy to recognize visually the main features of the flow, such as its vortex structures. We thus think that our model enabled the creation of reduced physical states that contain additional relevant information compared to a traditional down-sampling operation.

6.1.2 Perspectives

A major drawback of this method is its runtime performance. Indeed, we showed that our *ATO* model accelerated the reference simulation by about 20% on average in the different scenarios that we showcased. Even though our model improves the baseline and the other state-of-the-art

models, it still presents a big loss in quality compared to the reference. For that reason, we can wonder whether an artist or engineer that wishes to create a high-quality simulation would actually use our framework, or rather wait a little longer and use more resources to get a much better solution. Moreover, although our model accelerates the performance at inference time, it is quite heavy to train and we did not take the training time and resources into account when comparing performance with the high-resolution solver. Nonetheless, we believe that using a deep neural network model to discover a relevant reduced space for fluid simulation was an interesting first step, and that more work remains to be done if we want this idea to be implemented for real use cases.

First of all, our architecture was not optimized and some code optimization might lighten our computations. Secondly, we did not try for a more significant dimension reduction, i.e. having a latent space for example eight times smaller than the reference space, instead of four. This would allow for the production of a solution at a higher resolution than 256×256 and thus the application to more realistic scenarios. Finally, when we analyzed the performance of our model more deeply, we noticed that about half of the runtime (at training as well as inference) was taken by the solver step. In our experiments, we used the differentiable solver from the ϕ_{Flow} framework (itself using the ϕ_{ML} library [55]) but we did not investigate thoroughly its acceleration structures nor looked for potentially more efficient solvers, which might considerably improve the runtime performance of our *ATO* model.

If a faster and lighter architecture than ours is found to explore physical latent spaces in order to predict high-resolution simulations, an obvious follow-up would be the extension to 3D scenarios. In the current state of our model, solving the equations for such a scale was too computationally intensive and training our model for 8 solver steps would have taken several months, which is highly impractical. To extend our work to 3D, some inspiration could be taken from other fields such as biomedical imaging, in which people are used to treating some heavy 3D data [50, 52]. The extension of our model to 3D would also open the possibility to interact in real-time with a simulation, and thus let the user modify its physical properties. We could think of a use-case where the user would set up an initial coarse frame that is not too costly to make, control some simulation parameters, and then use our model to produce n frames in high resolution. The control parameters could for instance be different external forces, or the viscosity of the fluid. An interesting application would be to couple this with virtual reality, to enable the real-time production of high-resolution 3D fluid flows, while being able to shape the simulations in an interactive way.

6.2 Intrinsic SPH on surfaces

6.2.1 Contributions

In Chapter 5, we aimed at simulating fluids on surfaces at interactive speed, by proposing an intrinsic adaptation of the smoothed-particle hydrodynamics (SPH) method for arbitrary meshes. Since we wished to model fluid flows on 3D surfaces, we decided to modify the SPH algorithm by using *geodesic* distances and directions to gather the neighborhood of a particle. To do so, we extended the MMP algorithm to get the geodesic distance between any point on the surface to any other, at arbitrary positions. Furthermore, we proposed a *walk* algorithm that enables the intrinsic displacement of a particle on the surface. We update its barycentric coordinates thanks to its velocity and, in the case where its trajectory crosses an edge, we snap its position at the intersection. We then transform its velocity in the adjacent triangle and continue the walk until it finishes. Thanks to our intrinsic SPH simulation, we

were able to simulate fluids with various physical properties at interactive speeds, using tens of thousands of particles, on arbitrary surfaces, with the single constraint that one edge of the mesh could not be shared by more than two triangles. In this chapter, we showcased a basic implementation of SPH, but most features of this method are compatible with our intrinsic simulation. We implemented a few effects such as surface tension or droplets falling from the mesh onto another surface, and demonstrated our results on non-orientable and/or self-intersecting meshes such as the Klein bottle and the Moebius strip.

6.2.2 Perspectives

A first interesting avenue regarding this work is the study of the links between the simulation parameters and the geometrical properties of the mesh. More specifically, SPH importantly relies on its kernel radius, which both influences the speed of the computations and the stability of the simulation. In our case, it makes the simulation highly dependent on the length of the triangle edges, both for our neighborhood calculations and our walk algorithm. An analysis of the optimal relation between the edge length and the SPH kernel could be followed by an intrinsic triangulation, as presented in [43], in order to get a secondary and better-suited mesh on which to perform our intrinsic SPH simulation. This could greatly improve performance both in terms of runtime and quality of the results. It would also enable the mesh to have triangles that are more isotropic than they would originally be, which would make our neighborhood gathering more efficient and our walk algorithm more robust.

To further extend our intrinsic simulation framework, we could implement state-of-the-art SPH variants such as incompressibility, kernel reweighting, better boundary handling or multi-phase flows. These are all compatible with our neighborhood gathering and our use of geodesic distances. A more challenging task would be the addition of a small thickness to our simulation. In its current state, our framework already includes droplet simulation when the density of a particle goes over a threshold. It would add realism to actually be able to accumulate matter with a growing thickness, and then release some of it with droplets. To do so, we could for example add an artificial thickness when rendering the fluid. We could also decide that if an area has an average density above a certain value, we allow 3D SPH at a given radius around that area. Coupled with droplets, it would enable realistic simulation of thin fluids at interactive speeds within a simple framework. An interesting application of this, that would require quite some additional work, would be the simulation of condensation with droplet formation.

A more direct application of our work would be to use it to reproduce *dendritic paintings* as proposed by Canabal et al. in [25] (Figure 6.1 - left). In this paper, they present the simulation of dendritic patterns on a 2D regular grid. They couple a reaction-diffusion mechanism with the Lattice-Boltzmann Method (LBM) to create the patterns and to make some paint flow inside of them. We could extend their work to triangle meshes by using our intrinsic SPH simulation instead of their LBM. The least trivial part would be to find a way to link the particles to the reaction-diffusion happening on the mesh vertices. This could be done by creating some virtual sites and registering for instance the density of fluid on these sites when a particle goes through them. That way, we would be able to recreate for example some mocha diffusion art on 3D meshes, as shown on Figure 6.1 - right.

On the geometry processing side, some work can be done to improve our neighborhood computation. One limitation of our work for instance is the fact that we cannot set the size of a particle's neighborhood dynamically, because we need to pre-allocate the memory on GPU. Some cases can lead to scenarios where a particle actually has more neighbors than the maximum set amount, and the neighborhood becomes a bit arbitrary. It is unfortunately not

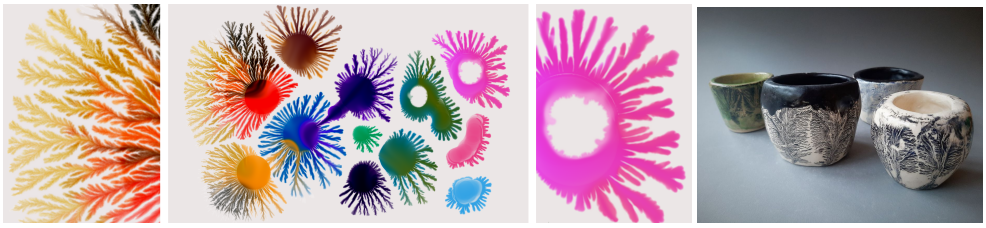

Figure 6.1: Results from the paper "Simulation of Dendritic Painting" [25] (left) and real mocha diffusion on pottery (right) from [96]

possible at the moment to allocate the memory dynamically, but we could however manage our memory usage more efficiently. Indeed, we could analyze more thoroughly the statistics of the particles' neighborhoods in order to allocate memory in a more relevant manner. Finally, part of our pre-processing steps are not parallelized yet.

Another interesting extension of our work would be to enable particles to cross corners, in order to allow for non-manifold meshes in which some triangles are connected by only one vertex. Finally, it would be useful to be able to interactively deform the mesh while the simulation is taking place. It would only require the new deformed mesh to be sent on GPU, and the triangles' neighborhoods to be computed again. Nevertheless, this cannot be done too often, as it would significantly slow down the performance. To circumvent this issue, we could recompute only the neighborhoods of the triangles that are within a certain radius around the ones that have been deformed.

6.3 Societal concerns

Working with computationally intensive simulations as well as deep learning models raises some ecological and thus sociological questions. The most obvious criticism relates to the very high consumption of resources induced by high-resolution and/or real-time simulations, as well as the use of deep neural networks. Although lots of works, like ours, narrowed their focus on improving the performance and thus usually decreasing the power and memory usage of such models, one could wonder how they are truly used. Indeed, lots of works in many fields focused on the now famous *rebound effect*, defined in [145] as follows: "The rebound effect deals with the fact that improvements in efficiency often lead to cost reductions that provide the possibility to buy more of the improved product or other products or services." A typical example of this effect is the fact that people use their car more frequently and for longer distances as technology makes them more efficient for fuel consumption. Closer to our subject, when new and usually more efficient graphics hardware is coming out, or when new algorithms or methods present a better runtime performance, a rebound effect also happens. This has been described as *Blinn's law* [109]: "As technology advances, rendering time remains constant." Indeed, it has been observed that the average rendering time for Pixar movies remained more or less constant over the last fifteen years, although technologies have significantly evolved since then [106].

Most scientific reviews keep measuring the quality of the proposed works with metrics that mostly highlight the realism of the results, and how much closer to a ground truth they are than previous works. One solution to overcome this rebound effect would thus be for the various scientific communities to commonly agree to change their appreciation criterias, by including – and insisting on – the power usage and the environmental and social impact of

Figure 6.2: Results from the paper "Lifted Curls: A Model for Tightly Coiled Hair Simulation" [132] published in 2023, comparing two different coiled hair simulations to real-world looks.

the reviewed work. In the current sociological context, we find it hard to keep selling the fact that we are able to produce simulations of ever increasing quality, especially since we already have beautiful photorealistic results as industry standards. We believe that research should be aligned with the important challenges that society is facing, climate change and the various societal crises being a major part of them.

Finally, most deep learning models require huge amounts of computational resources and data if one wants to reproduce the results presented by their authors. This poses several problems, the main ones being the negative impact on the environment and the unequal access to this technology. To tackle these issues, more and more engineers and researchers get interested in *frugal AI*, which consists in reducing the amount of resources used by deep neural networks. They propose multiple types of solutions, from changing the organization of data centers and hardware [111] to modifying the architecture and training procedure of neural networks [112].

In this manuscript we focused on the animation of non-human elements. A significant part of research in computer graphics however is dedicated to making virtual humans as physically realistic as possible. This goes from animating their gait or movements to simulating their skin, hair, bodies. To conclude this thesis, we would like to underline the importance of bias in computer graphics projects. Theodore Kim has addressed the question of racial bias when rendering human skin, or simulating human hair, in a talk at SIGGRAPH in 2021 [73]. In this talk, he showed that the lack of diversity when representing human skin was particularly appalling with the success of subsurface scattering models [63]. Subsurface scattering happens for every skin color, but to very different extents. In fact, the predominant effect for black skins is *not* subsurface scattering, contrary to white ones. Nevertheless, after the publication of this seminal paper in 2001, articles have been referring to their models as "skin models" instead of "white skin models" [139], which shows how computer graphics research – like most research fields – is centered on white people. Similar behaviors happen when it comes to simulating hair, with straight (or slightly curly) hair being the default representation [163] and kinky (i.e. extremely curly) hair (Figure 6.2) being almost inexistant in the literature before the years 2020 [132].

Furthermore, neural networks have gotten extremely popular for the general public since the rise of content generation with AI (text, image or video). The very essence of neural networks is to output results depending on the data they have seen during training, which is usually found on the internet or annotated by hand by humans. This means that socially biased training sets, as they almost all are, will produce biased outputs. For example, some works assess the racial biases [74] as well as the ones concerning sex and gender [38] in computer graphics research. They propose to change the way we simulate humans by using more universal models, and to change our default representations – usually white/physically

valid/thin/gender binary – to ones that represent the human diversity better. We also believe that it is our role as researchers to battle for less biased datasets, which would have a very direct impact on the public as they would lead to less biased data generation and thus representations.

Indeed, we highlighted all over this manuscript how important computer graphics research was for the various entertainment industries, themselves having a huge influence on society's representations. For that reason, we believe that researchers have a critical role to play in preventing the reproduction of people's biases, and in the creation of virtual worlds that equally represent all skin colors, hair types, genders and sexualities, as well as the tremendous diversity of body types, shapes and (dis)abilities that make the human race so beautiful.

Bibliography

[1] Raphaël Achddou. "Synthetic learning for neural image restoration methods". Theses. Institut Polytechnique de Paris, Mar. 2023. URL: https://theses.hal.science/tel-04164873.

[2] Haitham Afan et al. "Modeling the fluctuations of groundwater level by employing ensemble deep learning techniques". In: *Engineering Applications of Computational Fluid Mechanics* 15 (Sept. 2021), pp. 1420–1439. DOI: 10.1080/19942060.2021.1974093.

[3] Nadir Akinci et al. "Versatile Rigid-Fluid Coupling for Incompressible SPH". In: *ACM Trans. Graph.* 31.4 (July 2012), 62:1–62:8. ISSN: 0730-0301. DOI: 10.1145/2185520.2185558.

[4] Shunichi Amari. "A Theory of Adaptive Pattern Classifiers". In: *IEEE Transactions on Electronic Computers* EC-16.3 (1967), pp. 299–307. DOI: 10.1109/PGEC.1967.264666.

[5] Brandon Amos and J. Zico Kolter. "OptNet: Differentiable Optimization as a Layer in Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, June 2017, pp. 136–145. URL: https://proceedings.mlr.press/v70/amos17a.html.

[6] S. Auer and R. Westermann. "A Semi-Lagrangian Closest Point Method for Deforming Surfaces". In: *Computer Graphics Forum* 32.7 (2013), pp. 207–214. DOI: https://doi.org/10.1111/cgf.12228.

[7] S. Auer et al. "Real-Time Fluid Effects on Surfaces using the Closest Point Method". In: *Computer Graphics Forum* (2012). ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2012.03071.x.

[8] Filipe de Avila Belbute-Peres et al. "End-to-End Differentiable Physics for Learning and Control". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/842424a1d0595b76ec4fa03c46e8d755-Paper.pdf.

[9] Omri Azencot et al. "Functional thin films on surfaces". In: *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '15. Los Angeles, California: Association for Computing Machinery, 2015, pp. 137–146. ISBN: 9781450334969. DOI: 10.1145/2786784.2786793.

[10] Kai Bai et al. "Dynamic Upsampling of Smoke through Dictionary-Based Learning". In: *ACM Transactions on Graphics* 40.1 (Sept. 2020), 4:1–4:19. ISSN: 0730-0301. DOI: 10.1145/3412360.

[11] Stefan Band et al. “MLS Pressure Boundaries for Divergence-Free and Viscous SPH Fluids”. In: *Computers and Graphics* 76 (Aug. 2018). DOI: 10.1016/j.cag.2018.08.001.

[12] Stefan Band et al. “Pressure Boundaries for Implicit Incompressible SPH”. In: *ACM Transactions on Graphics* 37 (Feb. 2018), pp. 1–11. DOI: 10.1145/3180486.

[13] Yohai Bar-Sinai et al. “Learning data-driven discretizations for partial differential equations”. In: *Proceedings of the National Academy of Sciences* 116.31 (2019), pp. 15344–15349. DOI: 10.1073/pnas.1814058116.

[14] J. Bender and D. Koschier. “Divergence-Free SPH for Incompressible and Viscous Fluids”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (Mar. 2017), pp. 1193–1206. ISSN: 1077-2626. DOI: 10.1109/TVCG.2016.2578335.

[15] Jan Bender et al. “Implicit Frictional Boundary Handling for SPH”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.10 (2020), pp. 2982–2993. DOI: 10.1109/TVCG.2020.3004245.

[16] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. “A Neural Probabilistic Language Model”. In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[17] Anand Bharadwaj et al. “A discrete droplet method for modelling thin film flows”. In: *Applied Mathematical Modelling* 112 (2022), pp. 486–504. ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2022.08.001.

[18] Kaushik Bhattacharya et al. “Model Reduction and Neural Networks for Parametric PDEs”. In: *The SMAI journal of computational mathematics* 7 (2021), pp. 121–157. DOI: 10.5802/smai-jcm.74.

[19] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[20] J. Brackbill and H.M. Ruppel. “FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions”. In: *Journal of Computational Physics* 65 (Aug. 1986), pp. 314–343. DOI: 10.1016/0021-9991(86)90211-1.

[21] J.U. Brackbill, D.B. Kothe, and H.M. Ruppel. “FLIP: A Low-Dissipation, Particle-in-Cell Method for Fluid Flow”. In: *Computer Physics Communications* 48.1 (Jan. 1988), pp. 25–38. ISSN: 0010-4655. DOI: 10.1016/0010-4655(88)90020-3.

[22] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. “Message Passing Neural PDE Solvers”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=vSix3HPYKSU.

[23] Robert Edward Bridson. “Computational aspects of dynamic surfaces”. AAI3090563. PhD thesis. Stanford, CA, USA, 2003.

[24] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the National Academy of Sciences* 113.15 (2016), pp. 3932–3937.

[25] José A. Canabal et al. “Simulation of Dendritic Painting”. In: *Computer Graphics Forum* (2020). ISSN: 1467-8659. DOI: 10.1111/cgf.13955.

[26] Kathleen Champion et al. “Data-Driven Discovery of Coordinates and Governing Equations”. In: *Proceedings of the National Academy of Sciences* 116.45 (Nov. 2019), pp. 22445–22451. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1906995116.

[27] Jindong Chen and Yijie Han. "Shortest paths on a polyhedron". In: *Proceedings of the Sixth Annual Symposium on Computational Geometry*. SCG '90. Berkley, California, USA: Association for Computing Machinery, 1990, pp. 360–369. ISBN: 0897913620. DOI: 10.1145/98524.98601.

[28] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[29] Alexandre Joel Chorin. "The numerical solution of the Navier-Stokes equations for an incompressible fluid". In: *Bulletin of the American Mathematical Society* 73.6 (1967), pp. 928–931.

[30] Mengyu Chu and Nils Thuerey. "Data-Driven Synthesis of Smoke Flows with CNN-Based Feature Descriptors". In: *ACM Trans. Graph.* 36.4 (July 2017), 69:1–69:14. ISSN: 0730-0301. DOI: 10.1145/3072959.3073643.

[31] Keenan Crane, Mathieu Desbrun, and Peter Schröder. "Trivial connections on discrete surfaces". In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1525–1533.

[32] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. "Geodesics in heat: A new approach to computing distance based on heat flow". In: *ACM Trans. Graph.* 32.5 (Oct. 2013). ISSN: 0730-0301. DOI: 10.1145/2516971.2516977.

[33] James P Crutchfield and Bruce S McNamara. "Equations of motion from a data series". In: *Complex systems* 1.417-452 (1987), p. 121.

[34] Yitong Deng et al. "A moving eulerian-lagrangian particle method for thin film and foam simulation". In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530174.

[35] Mathieu Desbrun and Marie-Paule Cani. "Smoothed particles: a new paradigm for animating highly deformable bodies". In: *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*. Poitiers, France: Springer-Verlag, 1996, pp. 61–76. ISBN: 3211828850.

[36] *Diffusion Models vs. GANs vs. VAEs: Comparison of Deep Generative Models*. URL: https://pub.towardsai.net/diffusion-models-vs-gans-vs-vaes-comparison-of-deep-generative-models-67ab93e0d9ae.

[37] EW Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1.1 (1959), pp. 269–271.

[38] Ana Dodik et al. "Sex and Gender in the Computer Graphics Research Literature". In: *ACM SIGGRAPH 2022 Talks*. SIGGRAPH '22. Vancouver, BC, Canada: Association for Computing Machinery, 2022. ISBN: 9781450393713. DOI: 10.1145/3532836.3536227.

[39] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[40] Z. Fan et al. “Adapted unstructured LBM for flow simulation on curved surfaces”. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’05. Los Angeles, California: Association for Computing Machinery, 2005, pp. 245–254. ISBN: 1595931988. DOI: 10.1145/1073368.1073404.

[41] Kai Fukami, Koji Fukagata, and Kunihiko Taira. “Super-resolution reconstruction of turbulent flows with machine learning”. In: *Journal of Fluid Mechanics* 870 (2019), pp. 106–120. DOI: 10.1017/jfm.2019.238.

[42] Kai Fukami et al. “Sparse Identification of Nonlinear Dynamics with Low-Dimensionalized Flow Representations”. In: *Journal of Fluid Mechanics* 926 (Nov. 2021). ISSN: 0022-1120, 1469-7645. DOI: 10.1017/jfm.2021.697.

[43] Mark Gillespie, Nicholas Sharp, and Keenan Crane. “Integer coordinates for intrinsic geometry processing”. In: *ACM Trans. Graph.* 40.6 (Dec. 2021). ISSN: 0730-0301. DOI: 10.1145/3478513.3480522.

[44] Yoav Goldberg. “A primer on neural network models for natural language processing”. In: *J. Artif. Int. Res.* 57.1 (Sept. 2016), pp. 345–420. ISSN: 1076-9757.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[46] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[47] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. “Smoothed Particle Hydrodynamics on GPUs”. In: *Computer Graphics International* (Jan. 2007).

[48] Francis H Harlow. “The particle-in-cell method for numerical solution of problems in fluid dynamics”. In: (Mar. 1962). DOI: 10.2172/4769185.

[49] Francis H. Harlow and J. Eddie Welch. “Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface”. In: *Physics of Fluids* 8.12 (Dec. 1965), pp. 2182–2189. DOI: 10.1063/1.1761178.

[50] Ali Hatamizadeh et al. “UNETR: Transformers for 3D Medical Image Segmentation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2022, pp. 574–584.

[51] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385 [cs].

[52] Tobias Heimann and Hans-Peter Meinzer. “Statistical shape models for 3D medical image segmentation: A review”. In: *Medical Image Analysis* 13.4 (2009), pp. 543–563.

[53] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[54] Philipp Holl, Vladlen Koltun, and Nils Thuerey. “Learning to Control PDEs with Differentiable Physics”. In: *International Conference on Learning Representations (ICLR)* (2020).

[55] Philipp Holl and Nils Thuerey. “Phi-ML: Intuitive Scientific Computing with Dimension Types for Jax, PyTorch, TensorFlow and NumPy”. In: *Journal of Open Source Software* 9.95 (2024), p. 6171. DOI: 10.21105/joss.06171.

[56] Xiangyu Hu and Nikolaus Adams. "A multi-phase SPH method for macroscopic and mesoscopic". In: *Journal of Computational Physics* 213 (Apr. 2006), pp. 844–861. DOI: 10.1016/j.jcp.2005.09.001.

[57] Yuanming Hu et al. "DiffTaichi: Differentiable Programming for Physical Simulation". In: *International Conference on Learning Representations (ICLR)* (2020).

[58] Gao Huang et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

[59] Markus Ihmsen et al. "Boundary Handling and Adaptive Time-stepping for PCISPH". In: Jan. 2010, pp. 79–88. DOI: 10.2312/PE/vriphys/vriphys10/079-088.

[60] Markus Ihmsen et al. "Implicit Incompressible SPH". In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 426–435. DOI: 10.1109/TVCG.2013.105.

[61] Markus Ihmsen et al. "SPH Fluids in Computer Graphics". In: *Eurographics 2014 - State of the Art Reports*. Ed. by Sylvain Lefebvre and Michela Spagnuolo. The Eurographics Association, 2014. DOI: 10.2312/egst.20141034.

[62] Mike Innes et al. *A Differentiable Programming System to Bridge Machine Learning and Scientific Computing*. 2019. arXiv: 1907.07587 [cs.PL].

[63] Henrik Wann Jensen et al. "A practical model for subsurface light transport". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 511–518. ISBN: 158113374X. DOI: 10.1145/383259.383319.

[64] George Em Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[65] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4396–4405. DOI: 10.1109/CVPR.2019.00453.

[66] Tero Karras et al. *Alias-Free Generative Adversarial Networks*. 2021. arXiv: 2106.12423 [cs.CV].

[67] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: 1912.04958 [cs.CV].

[68] Enrico Puppo Keenan Crane Marco Livesu and Yipeng Qin. "A Survey of Algorithms for Geodesic Paths and Distances". In: *CoRR* abs/2007.10430 (2020). arXiv: 2007.10430. URL: https://arxiv.org/abs/2007.10430.

[69] Ioannis G Kevrekidis et al. "Equation-free, coarse-grained multiscale computation: Enabling mocroscopic simulators to perform system-level analysis". In: *Communications in Mathematical Sciences* 1.4 (2003), pp. 715–762.

[70] J. Kiefer and J. Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function". In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 462–466. DOI: 10.1214/aoms/1177729392.

[71] ByungMoon Kim et al. "FlowFixer: Using BFECC for Fluid Simulation". In: *Eurographics Conference on Natural Phenomena*. Dublin, Ireland: Eurographics Association, 2005, pp. 51–56. ISBN: 3-905673-29-0. DOI: 10.2312/NPH/NPH05/051-056.

[72] Byungsoo Kim et al. “Deep Fluids: A Generative Network for Parameterized Fluid Simulations”. In: *Computer Graphics Forum* (2019). ISSN: 1467-8659. DOI: 10.1111/cgf.13619.

[73] Theodore Kim. *Anti-Racist Graphics Research, talk at SIGGRAPH*. 2021. URL: https://www.youtube.com/watch?v=ROuE8xYLpX8.

[74] Theodore Kim et al. “Countering Racial Bias in Computer Graphics Research”. In: *ACM SIGGRAPH 2022 Talks*. SIGGRAPH ’22. Vancouver, BC, Canada: Association for Computing Machinery, 2022. ISBN: 9781450393713. DOI: 10.1145/3532836.3536263.

[75] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980 [cs].

[76] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014. arXiv: http://arxiv.org/abs/1312.6114v10 [stat.ML].

[77] Felix Knöppel et al. “Stripe patterns on surfaces”. In: *ACM Trans. Graph.* 34.4 (July 2015). ISSN: 0730-0301. DOI: 10.1145/2767000. URL: https://doi.org/10.1145/2767000.

[78] Dmitrii Kochkov et al. “Machine Learning–Accelerated Computational Fluid Dynamics”. In: *Proceedings of the National Academy of Sciences* 118.21 (May 2021). ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.2101784118.

[79] Dan Koschier and Jan Bender. “Density maps for improved SPH boundary handling”. In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA ’17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350914. DOI: 10.1145/3099564.3099565.

[80] Dan Koschier et al. “A Survey on SPH Methods in Computer Graphics”. In: *Computer Graphics Forum* 41.2 (2022), pp. 737–760. DOI: https://doi.org/10.1111/cgf.14508.

[81] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2012, pp. 1097–1105.

[82] Lubor Ladický et al. “Data-Driven Fluid Simulations Using Regression Forests”. In: *ACM Trans. Graph.* 34.6 (Oct. 2015), 199:1–199:9. ISSN: 0730-0301. DOI: 10.1145/2816795.2818129.

[83] Y. LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural Comput.* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[84] A.E. Lefohn et al. “Interactive deformation and visualization of level set surfaces using graphics hardware”. In: *IEEE Visualization, 2003. VIS 2003.* 2003, pp. 75–82. DOI: 10.1109/VISUAL.2003.1250357.

[85] Zijie Li and Amir Barati Farimani. “Graph neural network-accelerated Lagrangian fluid simulation”. In: *Comput. Graph.* 103.C (Apr. 2022), pp. 201–211. ISSN: 0097-8493. DOI: 10.1016/j.cag.2022.02.004.

[86] Zongyi Li et al. “Fourier Neural Operator for Parametric Partial Differential Equations”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=c8P9NQVtmnO.

[87] Junbang Liang, Ming C. Lin, and Vladlen Koltun. "Differentiable cloth simulation for inverse problems". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[88] M. B. Liu and G. R. Liu. "Smoothed Particle Hydrodynamics (SPH): an Overview and Recent Developments". In: *Archives of Computational Methods in Engineering* 17.1 (Mar. 2010), pp. 25–76. ISSN: 1886-1784. DOI: 10.1007/s11831-010-9040-7. URL: https://doi.org/10.1007/s11831-010-9040-7.

[89] Yong-Jin Liu, Qian-Yi Zhou, and Shi-Min Hu. "Handling degenerate cases in exact geodesic computation on triangle meshes". In: *Vis. Comput.* 23.9 (Aug. 2007), pp. 661–668. ISSN: 0178-2789. DOI: 10.1007/s00371-007-0136-5.

[90] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. "Simulating water and smoke with an octree data structure". In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 457–462. ISSN: 0730-0301. DOI: 10.1145/1015706.1015745.

[91] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. "Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics". In: *Nature Communications* 9.1 (Nov. 2018), p. 4950. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07210-0.

[92] Miles Macklin and Matthias Müller. "Position based fluids". In: *ACM Trans. Graph.* 32.4 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2461912.2461984.

[93] Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. "Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders". In: *Physics of Fluids* 33.3 (Mar. 2021), p. 037106. ISSN: 1070-6631. DOI: 10.1063/5.0039986.

[94] Aleka McAdams, Eftychios Sifakis, and Joseph Teran. "A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids." In: Jan. 2010, pp. 65–73. DOI: 10.2312/SCA/SCA10/065-073.

[95] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. "The Discrete Geodesic Problem". In: *SIAM Journal on Computing* 16.4 (1987), pp. 647–668. DOI: 10.1137/0216045.

[96] *Mocha diffusion example on pottery.* URL: https://glinaiwalek.pl/.

[97] Arvind Mohan et al. *Compressed Convolutional LSTM: An Efficient Deep Learning framework to Model High Fidelity 3D Turbulence*. 2019. arXiv: 1903.00033 [physics.flu-dyn].

[98] J.J. Monaghan. "Smoothed particle hydrodynamics." In: *Annual Review of Astronomy and Astrophysics* 30 (Jan. 1992), pp. 543–574. DOI: 10.1146/annurev.aa.30.090192.002551.

[99] D. Morgenroth et al. "Efficient 2D Simulation on Moving 3D Surfaces". In: *Computer Graphics Forum* 39.8 (2020), pp. 27–38. DOI: https://doi.org/10.1111/cgf.14098.

[100] Jeremy Morton et al. "Deep dynamical modeling and control of unsteady fluid flows". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 9278–9288.

[101] 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*. SCA '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 154–159. ISBN: 1-58113-659-5.

[102] Matthias Müller et al. “Position based dynamics”. In: *J. Vis. Comun. Image Represent.* 18.2 (Apr. 2007), pp. 109–118. ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2007.01.005.

[103] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[104] Ken Museth. “VDB: High-resolution sparse volumes with dynamic topology”. In: *ACM Trans. Graph.* 32.3 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2487228.2487235.

[105] Patrick Neill, Ron Metoyer, and Eugene Zhang. “Fluid flow on interacting deformable surfaces”. In: *ACM SIGGRAPH 2007 Posters*. SIGGRAPH ’07. San Diego, California: Association for Computing Machinery, 2007, 57–es. ISBN: 9781450318280. DOI: 10.1145/1280720.1280783.

[106] Alec Nevala-Lee. *Blinn’s Law and the paradox of efficiency*. URL: https://nevalalee.wordpress.com/2011/08/09/blinns-law-and-the-paradox-of-efficiency/.

[107] Young Jin Oh and In-Kwon Lee. “Two-step Temporal Interpolation Network Using Forward Advection for Efficient Smoke Simulation”. In: *Computer Graphics Forum* 40.2 (May 2021), pp. 355–365. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.142638.

[108] Or Patashnik et al. “StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 2085–2094.

[109] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016. ISBN: 0128006455.

[110] Konrad Polthier and Markus Schmies. “Straightest geodesics on polyhedral surfaces”. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH ’06. Boston, Massachusetts: Association for Computing Machinery, 2006, pp. 30–38. ISBN: 1595933646. DOI: 10.1145/1185657.1185664. URL: https://doi.org/10.1145/1185657.1185664.

[111] *Qarnot Computing*. URL: https://qarnot.com/fr.

[112] Aël Quélennec et al. “Towards On-device Learning on the Edge: Ways to Select Neurons to Update under a Budget Constraint”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2024, pp. 685–694. URL: https://openaccess.thecvf.com/content/WACV2024W/SCIoT/papers/Quelennec_Towards_On-Device_Learning_on_the_Edge_Ways_To_Select_Neurons_WACVW_2024_paper.pdf.

[113] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 18–24 Jul 2021, pp. 8748–8763. URL: https://proceedings.mlr.press/v139/radford21a.html.

[114] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[115] Karthik Raveendran, Chris Wojtan, and Greg Turk. “Hybrid Smoothed Particle Hydrodynamics”. In: *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’11. New York, NY, USA: ACM, 2011, pp. 33–42. ISBN: 978-1-4503-0923-3. DOI: 10.1145/2019406.2019411.

[116] Nicolas Ray et al. “N-symmetry direction field design”. In: *ACM Trans. Graph.* 27.2 (May 2008). ISSN: 0730-0301. DOI: 10.1145/1356682.1356683.

[117] Bo Ren et al. “Real-Time High-Fidelity Surface Flow Simulation”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.8 (2018), pp. 2411–2423. DOI: 10.1109/TVCG.2017.2720672.

[118] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586.

[119] Robin Rombach et al. “High-Resolution Image Synthesis With Latent Diffusion Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 10684–10695.

[120] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2015, pp. 234–241.

[121] Frank Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65 (1958), pp. 386–408.

[122] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022. arXiv: 2205.11487 [cs.CV].

[123] Alvaro Sanchez-Gonzalez et al. *Graph networks as learnable physics engines for inference and control*. 2018. arXiv: 1806.01242 [cs.LG].

[124] Alvaro Sanchez-Gonzalez et al. “Learning to simulate complex physics with graph networks”. In: *Proceedings of the 37th International Conference on Machine Learning*. ICML’20. JMLR.org, 2020.

[125] Connor Schenck and Dieter Fox. “SPNets: Differentiable Fluid Dynamics for Deep Neural Networks”. In: *ArXiv* abs/1806.06094 (2018). URL: https://api.semanticscholar.org/CorpusID:49207686.

[126] Samuel S. Schoenholz and Ekin D. Cubuk. *JAX, M.D.: A Framework for Differentiable Physics*. 2020. arXiv: 1912.04232 [physics.comp-ph].

[127] Andrew Selle et al. “An Unconditionally Stable MacCormack Method”. In: *Journal of Scientific Computing* 35.2-3 (June 2008), pp. 350–371. ISSN: 0885-7474, 1573-7691. DOI: 10.1007/s10915-007-9166-4.

[128] Rajsekhar Setaluri et al. “SPGrid: a sparse paged grid structure applied to adaptive smoke simulation”. In: *ACM Trans. Graph.* 33.6 (Nov. 2014). ISSN: 0730-0301. DOI: 10.1145/2661229.2661269.

[129] Nicholas Sharp and Keenan Crane. “You can find geodesic paths in triangle meshes by just flipping edges”. In: *ACM Trans. Graph.* 39.6 (Nov. 2020). ISSN: 0730-0301. DOI: 10.1145/3414685.3417839.

[130] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. “Navigating intrinsic triangulations”. In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3322979.

[131] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. “The Vector Heat Method”. In: *ACM Trans. Graph.* 38.3 (June 2019). ISSN: 0730-0301. DOI: 10.1145/3243651.

[132] Alvin Shi et al. “Lifted Curls: A Model for Tightly Coiled Hair Simulation”. In: *Proc. ACM Comput. Graph. Interact. Tech.* 6.3 (2023). DOI: 10.1145/3606920.

[133] Lin Shi and Yizhou Yu. "Inviscid and incompressible fluid simulation on triangle meshes". In: *Computer Animation and Virtual Worlds* 15.3-4 (2004), pp. 173–181. DOI: https://doi.org/10.1002/cav.19.

[134] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[135] Justin Sirignano, Jonathan F. MacArt, and Jonathan B. Freund. "DPM: A Deep Learning PDE Augmentation Method with Application to Large-Eddy Simulation". In: *Journal of Computational Physics* 423 (Dec. 2020), p. 109811. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2020.109811.

[136] B. Solenthaler and R. Pajarola. "Density contrast SPH interfaces". In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Dublin, Ireland: Eurographics Association, 2008, pp. 211–218. ISBN: 9783905674101.

[137] B. Solenthaler and R. Pajarola. "Predictive-corrective Incompressible SPH". In: *ACM Trans. Graph.* 28.3 (July 2009), 40:1–40:6. ISSN: 0730-0301. DOI: 10.1145/1531326.1531346.

[138] Kim Stachenfeld et al. "Learned Simulators for Turbulence". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=msRBojTz-Nh.

[139] Jos Stam. "An Illumination Model for a Skin Layer Bounded by Rough Surfaces". In: *Rendering Techniques 2001*. Ed. by Steven J. Gortler and Karol Myszkowski. Vienna: Springer Vienna, 2001, pp. 39–52. ISBN: 978-3-7091-6242-2.

[140] Jos Stam. "Stable Fluids". In: *SIGGRAPH '99*. ACM, 1999, pp. 121–128. ISBN: 0-201-48560-5. DOI: 10.1145/311535.311548.

[141] Oded Stein et al. "A Simple Discretization of the Vector Dirichlet Energy". In: *Computer Graphics Forum* 39.5 (2020). DOI: 10.1111/cgf.14070.

[142] Alexey Stomakhin et al. "A Material Point Method for Snow Simulation". In: *ACM Trans. Graph.* 32.4 (July 2013), 102:1–102:10. ISSN: 0730-0301. DOI: 10.1145/2461912.2461948.

[143] Vitaly Surazhsky et al. "Fast exact and approximate geodesics on meshes". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 553–560. ISSN: 0730-0301. DOI: 10.1145/1073204.1073228.

[144] Andre Tampubolon et al. "Multi-species simulation of porous sand and water mixtures". In: *ACM Transactions on Graphics* 36 (July 2017), pp. 1–11. DOI: 10.1145/3072959.3073651.

[145] Joan Thiesen et al. "Rebound effects of price differences". In: *The International Journal of Life Cycle Assessment* 13 (Mar. 2008), pp. 104–114. DOI: 10.1065/lca2006.12.297.

[146] Nils Thuerey et al. *Physics-based Deep Learning*. WWW, 2021. URL: https://physicsbaseddeeplearning.org.

[147] Jonathan Tompson et al. "Accelerating eulerian fluid simulation with convolutional networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3424–3433.

[148] Marc Toussaint et al. “Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 6231–6235. DOI: 10.24963/ijcai.2019/869. URL: https://doi.org/10.24963/ijcai.2019/869.

[149] Evgenii Tumanov, Dmitry Korobchenko, and Nuttapong Chentanez. “Data-Driven Particle-Based Liquid Simulation with Deep Learning Utilizing Sub-Pixel Convolution”. In: *Proc. ACM Comput. Graph. Interact. Tech.* 4.1 (Apr. 2021). DOI: 10.1145/3451261.

[150] Kiwon Um, Xiangyu Hu, and Nils Thuerey. “Liquid Splash Modeling with Neural Networks”. In: *Computer Graphics Forum* 37.8 (Dec. 2018), pp. 171–182. ISSN: 1467-8659. DOI: 10.1111/cgf.13522.

[151] Kiwon Um et al. “Solver-in-the-loop: learning from differentiable physics to interact with iterative PDE-solvers”. In: *Advances in Neural Information Processing Systems* 33 (2020). arXiv: 2007.00016.

[152] Benjamin Ummenhofer et al. “Lagrangian Fluid Simulation with Continuous Convolutions”. In: *International Conference on Learning Representations*. 2020. URL: https://api.semanticscholar.org/CorpusID:211165482.

[153] Orestis Vantzos, Saar Raz, and Mirela Ben-Chen. “Real-Time Viscous Thin Films”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275086.

[154] Orestis Vantzos et al. “Functional Thin Films on Surfaces”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (2017), pp. 1179–1192. DOI: 10.1109/TVCG.2016.2605083.

[155] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[156] Mengdi Wang et al. “Thin-film smoothed particle hydrodynamics fluid”. In: *ACM Trans. Graph.* 40.4 (July 2021). ISSN: 0730-0301. DOI: 10.1145/3450626.3459864.

[157] Rui Wang et al. “Towards Physics-Informed Deep Learning for Turbulent Flow Prediction”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 1457–1466. ISBN: 978-1-4503-7998-4. DOI: 10.1145/3394486.3403198.

[158] Wujie Wang, Simon Axelrod, and Rafael Gómez-Bombarelli. *Differentiable Molecular Simulations for Control and Learning*. 2020. arXiv: 2003.00868 [physics.comp-ph].

[159] S. Wiewel et al. “Latent Space Subdivision: Stable and Controllable Time Predictions for Fluid Flow”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’20. Goslar, DEU: Eurographics Association, Oct. 2020, pp. 1–11. DOI: 10.1111/cgf.14097.

[160] Xiangyun Xiao, Cheng Yang, and Xubo Yang. “Adaptive Learning-Based Projection Method for Smoke Simulation”. In: *Computer Animation and Virtual Worlds* 29.3-4 (May 2018), e1837. ISSN: 1546-427X. DOI: 10.1002/cav.1837.

[161] You Xie et al. “tempoGAN: a temporally coherent, volumetric GAN for super-resolution fluid flow”. In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201304.

[162] Bowen Yang et al. “Real-Time Fluid Simulation on the Surface of a Sphere”. In: *Proc. ACM Comput. Graph. Interact. Tech.* 2.1 (June 2019). DOI: 10.1145/3320285.

[163] Cem Yuksel, Scott Schaefer, and John Keyser. “Hair Meshes”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2009)* 28.5 (2009), 166:1–166:7. DOI: 10.1145/1661412.1618512.

[164] Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski. “An Advection-Reflection Solver for Detail-Preserving Fluid Simulation”. In: *ACM Trans. Graph.* 37.4 (July 2018), 85:1–85:8. ISSN: 0730-0301. DOI: 10.1145/3197517.3201324.

[165] Yihao Zhao, Ruihai Wu, and Hao Dong. “Unpaired Image-to-Image Translation Using Adversarial Consistency Loss”. In: Nov. 2020, pp. 800–815. ISBN: 978-3-030-58544-0. DOI: 10.1007/978-3-030-58545-7_46.

[166] Yongning Zhu and Robert Bridson. “Animating Sand As a Fluid”. In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 965–972. ISSN: 0730-0301. DOI: 10.1145/1073204.1073298.

[167] Jiawei Zhuang et al. “Learned Discretizations for Passive Scalar Advection in a Two-Dimensional Turbulent Flow”. In: *Physical Review Fluids* 6.6 (June 2021). DOI: 10.1103/PhysRevFluids.6.064605.

ECOLE
DOCTORALE

Titre : Réduction de dimension pour la simulation et l'animation de fluides

Mots clés : Informatique graphique, Simulation de fluides, Réduction de dimension, Géométrie intrinsèque, Apprentissage profond.

Résumé : Malgré les améliorations considérables des performances du matériel graphique ainsi que les avancées algorithmiques majeures depuis le début des années 2000, certains phénomènes naturels restent extrêmement coûteux à simuler. Par exemple, plusieurs pistes ont été proposées pour améliorer les performances des simulations de fluides, qui sont animées par la résolution d'équations différentielles partielles (EDP), plus particulièrement les équations hautement non linéaires de Navier-Stokes.
Dans cette thèse, nous explorons d'abord l'utilisation de l'apprentissage profond pour créer un espace réduit dans lequel un solveur peut opérer à moindre coût, tout en produisant des solutions de haute qualité. Nous proposons un modèle qui permet la simulation d'écoulements turbulents à une résolution quatre fois supérieure à celle de l'entrée dans chaque dimension, avec des performances d'exécution améliorées par rapport à un solveur haute résolution.
Ensuite, nous utilisons les contributions sur les opérateurs intrinsèques pour simuler des fluides sur des surfaces 3D avec des coûts réduits. Nous nous concentrons sur le modèle *smoothed-particle hydrodynamics* (SPH) que nous adaptons aux surfaces 3D, en rassemblant les voisinages des particules grâce aux géodésiques de plus court chemin, et en déplaçant ces particules de manière intrinsèque sur la surface. Tout ceci est simple à mettre en œuvre sur le GPU, ce qui permet la simulation de dizaines de milliers de particules sur différents maillages triangulaires à une vitesse interactive.

Title : Dimension reduction for fluid simulation and animation

Keywords : Computer graphics, Fluid simulation, Dimension reduction, Intrinsic geometry, Deep Learning.

Abstract : Despite tremendous improvements in graphics hardware performance as well as key algorithmic advancements since the beginning of the years 2000, some natural phenomena remain extremely costly to simulate. For instance, several tracks have been proposed to improve the performance of fluid simulations, that are animated by solving partial differential equations (PDE), more specifically the highly non-linear Navier-Stokes equations.
In this thesis, we first explore the use of deep learning to create a reduced space in which a solver can operate with lower costs, while still outputting high-quality solutions. We propose a model that enables the simulation of turbulent flows at a resolution four times higher than that of the given input in each dimension, with improved runtime performance compared to a high-resolution solver.
Secondly, we use the contributions on intrinsic operators for simulating fluids on 3D surfaces with reduced costs. We focus on the smoothed-particle hydrodynamics (SPH) model that we adapt to 3D surfaces, by gathering the particles' neighborhoods thanks to shortest-path geodesics, and by displacing such particles in an intrinsic manner on the surface. All of this is straightforward to implement on the GPU, enabling the simulation of tens of thousands of particles on various triangle meshes at interactive speed.

Institut Polytechnique de Paris
91120 Palaiseau, France