Diss. ETH No. 27198

Data-Driven Methods for Artist-Directed Fluid Simulations

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

Byungsoo Kim

MSc ETH in Computer Science, ETH Zurich, Switzerland

born on 08.06.1987

citizen of Republic of Korea

accepted on the recommendation of

Prof. Dr. Markus Gross, examiner

Dr. Barbara Solenthaler, co-examiner

Prof. Dr. Doug L. James, co-examiner

2020

Abstract

Fluid phenomena are ubiquitous to our world experience: winds swooshing through trembling leaves, turbulent water streams running down a river, and cellular patterns generated from wrinkled flames are some few examples. These complex phenomena capture our attention and awe due to the beautifully materialized complex patterns and become crucial elements to artistically support storytelling. In virtual environments, however, sophisticated manipulation of animated flow structures is still a burdensome task.

Given the amount of available fluid simulation data, data-driven approaches have emerged as attractive solutions. Hence, this thesis aims to provide a novel data-driven framework in which artists can freely interact and manipulate fluid simulations to achieve a desired style or to follow an input sketch, without the burden of tuning a vast number of physical parameters or long waiting times.

Aiming to improve the usually slow computational time of simulations, we present the first generative deep learning architecture that successfully synthesizes plausible and divergence-free 2D and 3D fluid simulation velocities from a set of reduced parameters. Our results show that generative neural networks can construct a wide variety of fluid behaviors, from turbulent smoke to viscous liquids, that closely match the input training data.

Neural Style Transfer is a popular technique for artistically manipulating images and videos. Hence, we introduce the first Transport-based Neural Style Transfer algorithm for smoke simulations. Our method facilitates automatic instantiation of a vast set of motifs through semantic transfer, enabling novel artistic manipulations for fluid simulation data. Additionally, the proposed method successfully synthesizes various styles of input images, ranging from artistic to photorealistic examples.

We then extend our previous Transport-based neural flow stylization to a Lagrangian framework, which demonstrates benefits with respect to quality, performance, and art-directability. A key property of our approach is that it is not restricted to any particular fluid solver type. This generality of our method facilitates seamless integration of neural style transfer into existing content production workflows.

Finally, we propose the first method for reconstructing 3D smoke densities from 2D artist sketches, which potentially represents the first step towards bridging the gap between early-stage prototyping of smoke keyframes and visual realization. We proposed a CNN for computing density refinements, a differentiable sketch renderer integrated into the end-to-end training, and a set of loss functions designed explicitly for the sketch-to-density problem.

Zusammenfassung

Naturphänomene wie Flüssigkeiten und Rauch sind allgegenwärtig: Winde, die durch Blätter wehen, turbulente Wassermassen, die einen Fluss hinunter strömen, und zellenähnliche Muster, welche von flackernden Flammen erzeugt werden, sind nur einige Beispiele. Diese komplexen und visuell spektakulären Phänomene sind nicht nur beeindruckend, sondern werden zu entscheidenden Elementen um eine Geschichte bildnerisch zu erzählen und künstlerisch zu unterstützen. In solchen virtuellen Umgebungen ist jedoch die Kontrolle dieser Flüssigkeiten und deren feiner Strukturen immer noch eine zeitintensive und aufwändige Aufgabe.

Angesichts der Menge der verfügbaren Daten von Strömungssimulationen haben sich Daten-gesteuerte Ansätze als attraktive Lösung herausgestellt. Ziel dieser Arbeit ist es daher, neuartige Daten-gesteuerte Methoden und Algorithmen bereitzustellen, in dem Künstler mit Flüssigkeitssimulationen interagieren und diese manipulieren können. Dies ermöglicht es, eine gewünschte Form oder Stil zu erzielen, ohne dass eine hohe Anzahl an physikalischen Variablen getestet werden muss und lange Wartezeiten entstehen.

Um die normalerweise langsame Berechnungszeit von Simulationen zu verbessern, präsentieren wir die erste generative Deep Learning Architektur, die plausible und divergenzfreie 2D- und 3D-Geschwindigkeitsfelder von Strömungen mit einer Reihe reduzierter Parameter synthetisiert. Unsere Ergebnisse zeigen, dass generative neuronale Netze eine Vielzahl von Flüssigkeitsverhalten nachbilden können, von turbulentem Rauch bis zu viskosen Flüssigkeiten, die stark mit den eingegebenen Trainingsdaten übereinstimmen.

Neural Style Transfer ist eine beliebte Technik zur künstlerischen Bearbeitung von Bildern und Videos. Daher stellen wir den ersten transport-basierten Neural Style Transfer-Algorithmus für Rauchsimulationen vor. Unsere Methode ermöglicht die automatische Instanziierung einer Vielzahl von Motiven durch semantischen Transfer und ermöglicht neuartige künstlerische Manipulationen von Strömungssimulationen. Darüber hinaus synthetisiert das vorgeschlagene Verfahren erfolgreich verschiedene Arten von Eingabebildern, die von künstlerischen bis zu fotorealistischen Beispielen reichen.

Anschließend erweitern wir unsere bisherige transportbasierte neuronale Strömungsstilisierung auf ein Lagrange Framework, das Vorteile in Bezug auf

Qualität, Laufzeit und künstlerische Kontrolle hat. Eine Schlüsseleigenschaft unseres Ansatzes ist, dass er nicht auf ein bestimmtes Diskretisierungsmodell und Drucklösertyp beschränkt ist. Diese Allgemeingültigkeit unserer Methode ermöglicht eine problemlose Integration der Methode in existierende Arbeitsprozesse und Spezialeffekte.

Abschliessend schlagen wir die erste Methode zur Rekonstruktion von 3D-Rauchdichten aus 2D-Skizzen von Künstlern vor, um die Brücke zwischen dem frühen Prototyping von Rauch Keyframes und der visuellen Realisierung in 3D zu ermöglichen. Wir präsentieren ein CNN-basiertes Netzwerk zur Berechnung der Dichte und Verfeinerung davon, einen differenzierbaren Sketch-Renderer der in das End-to-End-Training integriert ist, und eine Reihe von Kostenfunktionen, welche explizit für das Sketch-to-Density-Problem entwickelt wurden.

Acknowledgments

Above all, I would like to thank my advisor, Prof. Markus Gross, who allowed me to work at the Computer Graphics Lab, an excellent research environment. He also gave me an ideal opportunity to work at Disney Research Studio to render the research idea on a commercial scale. He provided me those places to run around without a leash but with trust, firm support, and constant attention during my Ph.D. Furthermore, his leadership and flexibility were also a dear lesson in my life.

I was indeed fortunate enough to work with my co-advisor, Dr. Barbara Solenthaler. She had also given me precious chances to grow up even before I started my Master's program. She could envision many machine learning ideas for fluid simulations and gratefully invited me to work with her to build them into this thesis. As one of the first members of the Simulation and Animation Group, I was able to closely learn a lot from her invaluable experience and warm guidance that held me not to falter during my Ph.D. I'm very thankful for having such a great advisor like her.

I would also like to thank my close collaborators for their advice and productive discussions. Luckily I was able to meet two distinguished researchers Prof. Nils Thuerey and Prof. Theodore Kim, during my first year of Ph.D. I sincerely thank both for steering me throughout our early work. They were always available to elucidate my issues and pointed me in the right direction from the dark. I would like to thank Dr. Guillaume Cordonnier and Xingchang Huang. I'm glad that we made a great work during this mad COVID-19 time. It was definitely anything but easy. Special thanks go to Fraser Rothnie for his artistic contributions.

I would like to thank Prof. Doug James, who graciously accepted to be a committee member for my Ph.D. examination.

I truly thank my dear colleagues at CGL and IGL for having our happy lunch + coffee breaks. I will never miss our CGL cuddling retreats as well. I would also like to thank my friends for their mental supports.

I am deeply thankful to my family. I would like to take this moment to say I love you, mom and dad, my wife Hyun Min, my son Jay, and my unborn daughter Jenny.

Finally, I would like to thank Dr. Vinicius C. Azevedo. He is more than a mentor to me. It will take up too much space to show how much I appreciate him. The last four years with you during my Ph.D. was like a gift to me. I have learned from you how to live a better life as a better person. Thanks, Vman.

The work was supported by the Swiss National Science Foundation under Grant No.: 200021_168997.

"It's a common misconception that visual effects are about simulating reality.
They're not. Reality is boring. Visual effects are about simulating something dramatic."

- Jonathan Cohen, Rhythm & Hues

Contents

List of Figures

List of Algorithms

List of Tables

CHAPTER 1

Introduction

Fluid phenomena are ubiquitous to our world experience: winds swooshing through trembling leaves, turbulent water streams running down a river, and cellular patterns generated from wrinkled flames are some few examples. These complex phenomena capture our attention and awe due to the beautifully materialized complex patterns and become crucial elements to artistically support storytelling. Thus, for the last 30 years, several works [Stam, 1999; Müller et al., 2003; Zhu and Bridson, 2005] in Computer Graphics have successfully focused on plausibly simulating flows in virtual environments.

However, artistically manipulating animated flow structures is still a burdensome task. In movies, flow features often have to be precisely guided to convey a dramatic effect or aesthetically compose a shot. Current workflow pipelines for controlling fluids are based on laborious hand-tuning of a myriad of parameters, which usually require specific knowledge of the underlying physics and algorithms used for numerical approximation. Due to the complexity of the underlying mathematical model, achieving a specific outcome can become counter-intuitive, and large-scale richly detailed fluid simulations are computationally expensive. These factors lead to a tedious and time-consuming artistic authoring process.

Meanwhile, machine learning techniques have become popular in recent years due to numerous algorithmic advances and the accessibility of computational resources. These, in particular, made a considerable revolution in semantic analysis [Simonyan and Zisserman, 2015] and generative tasks [Isola et al., 2017]. Accordingly, they have been adopted for many applications in

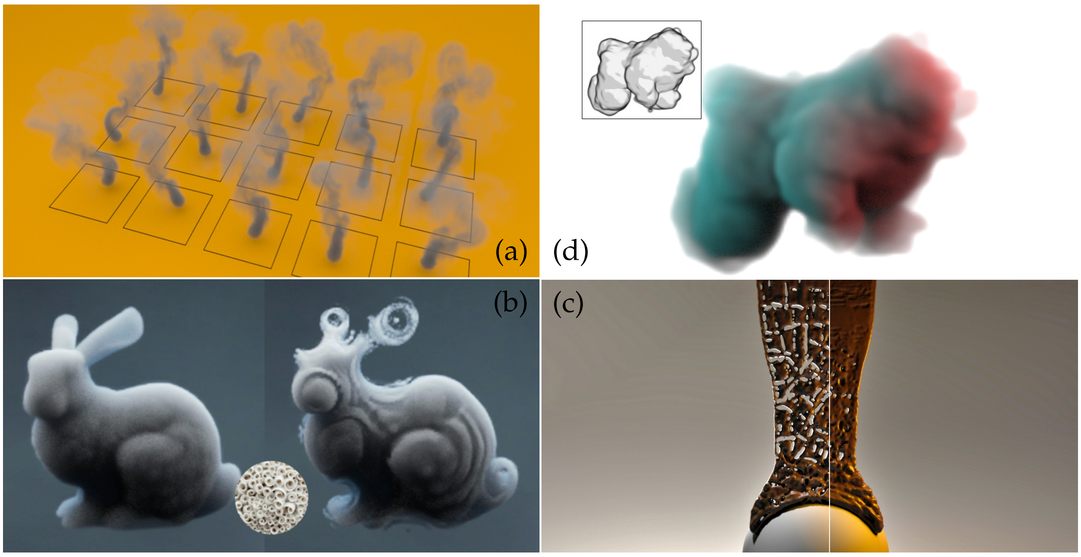


Figure 1.1: *Target applications of this thesis. We provide a powerful deep generative model spanning a vast subspace of fluids (a), physically-inspired neural flow stylization for both Eulerian (b) and Lagrangian (c) fluid representations, and the first sketch-to-density authoring tool (d).*

Computer Graphics, such as generating terrains [Guérin et al., 2017], high-resolution faces synthesis [Karras et al., 2018], imitating character's acrobatic dynamics [Peng et al., 2018], neural style transfer [Gatys et al., 2016a], video synthesis from audio for lip sync [Suwajanakorn et al., 2017], and cloud rendering [Kallweit et al., 2017]. In fluid simulation, machine learning techniques have been used to replace [Ladický et al., 2015], speed up [Tompson et al., 2019], or enhance existing solvers [Xie et al., 2018].

Hence, this thesis goal is to develop novel data-driven methods that support authoring fluid simulations. Thus, we developed tools to cope with the challenges of performance and physical plausibility (Chapter 3), image-based style transfer for volumetric grids (Chapter 4) and particle-based simulations (Chapter 5), and prototyping of simulations based on sketch inputs provided by artists (Chapter 6). The ultimate goal of this thesis is to provide a framework in which artists can freely interact and manipulate fluid simulations to achieve a desired style or to follow an input sketch, without the burden of tuning a vast number of physical parameters or long waiting times. In the following sections, we will further comment on the challenges of data-driven and artistic authoring of fluid simulations.

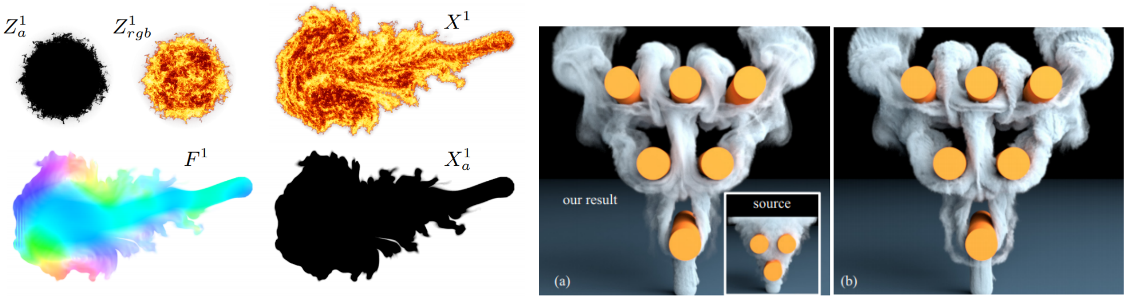


Figure 1.2: *Limitation of existing flow stylization approaches. Patch-based texture synthesis methods (left, [Jamriška et al., 2015]) allow appearance transfer (X^1) from a source texture (Z^1_{rgb}) to a target mask (X^1_a) in 2D only, and velocity synthesis methods (right, [Sato et al., 2018a]) merely focus on enhancing small-scale turbulence details.*

1.1 Data-Driven Fluid Simulations

The constant interplay between artists and digital content requires tools to change the scene setup iteratively, and computationally efficient simulations are desired to avoid long waiting times. Efficiency is even more critical for physically-based simulations in short animated TV productions, where simulated effects must be ready in instant time and at a low budget.

Given the amount of available fluid simulation data, data-driven approaches have emerged as attractive solutions. Subspace solvers [Treuille et al., 2006], fluid re-simulators [Kim and Delaney, 2013] and basis compressors [Demby Jones et al., 2016] are examples of recent efforts in this direction. However, these methods usually represent fluids using linear basis functions, e.g., constructed via Singular Value Decomposition (SVD), which are less efficient than their non-linear counterparts. Chapter 3 investigates deep generative models implemented by convolutional neural networks (CNNs) to show promise for representing data in reduced dimensions thanks to their capability to tailor non-linear functions to input data.

1.2 Artistic Fluids Authoring

Flow Stylization Post-processing methods for fluids aim at enabling detailed feature control by patch-based texture and velocity synthesis. While current patch-based techniques focus on controlling structural patterns [Ma et al., 2009; Jamriška et al., 2015], they are limited to 2D flows. Velocity synthesis approaches allow augmentation of coarse simulations with turbulent

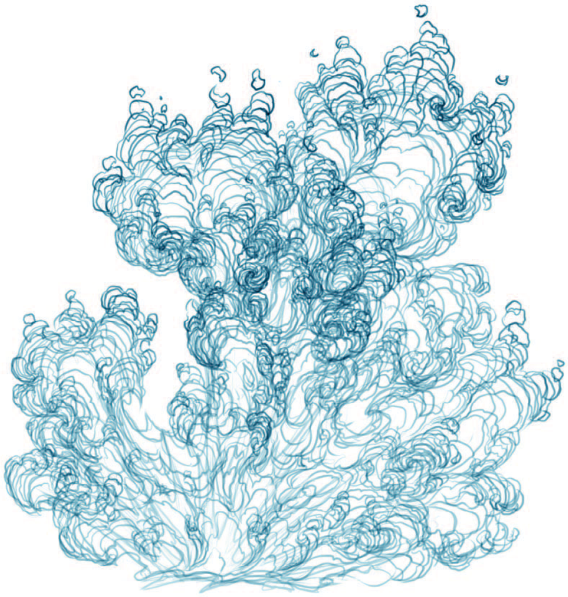


Figure 1.3: *Artist sketches of an explosion superimposed over time [Gilland, 2012]. To the best of our knowledge, no existing method allows artists to construct 3D smoke keyframes straight from 2D sketches.*

structures [Kim et al., 2008b; Sato et al., 2018a], but cannot capture the full spectrum of different styles and complex semantics. In Chapter 4 and Chapter 5, we examine the power of the recent neural algorithm to support artistic manipulations of flow data enabling multi-level control of flow features with automatic instantiation of patterns.

Sketch-Based Keyframe Construction At the core of creative workflows, digital artists start with concept sketches and a storyboard indicating the motion [Gilland, 2012] and manually generate individual keyframes representing the fluid at specific time instances. They are often represented as volumetric density fields [Pan and Manocha, 2016], but liquid surface information [Nielsen et al., 2011] and mesh boundaries [Raveendran et al., 2012] have also been previously employed. Animations are then generated by either integrating the keyframes into a physics solver with artificial forces attracting the smoke to the target shape [Thürey et al., 2006; Fattal and Lischinski, 2004; Shi and Yu, 2005b] or using gradient-based optimization techniques such as the adjoint method [Treuille et al., 2003; McNamara et al., 2004]. Reproducing these sketched keyframes as 3D density clouds that capture realistic flow details is highly non-trivial and remains a manual and time-consuming process. To our knowledge, no method exists that generates 3D reconstructions of fluids from 2D artist sketches. Thus, in Chapter 6, we study a data-guided approach to compute a 3D smoke density field directly from a set of 2D artist sketches, bridging the gap between early-stage prototyping of smoke keyframes and visual realization.

1.3 Contributions

In the following, we list the main contributions of the work presented in this thesis:

- We present a *novel generative model (Deep Fluids) to synthesize fluid simulations* from a set of reduced parameters. A convolutional neural network is trained on a collection of discrete, parameterizable fluid simulation velocity fields. Due to the capability of deep learning architectures to learn representative features of the data, our generative model is able to accurately approximate the training dataset, while providing plausible interpolated in-betweens. The proposed generative model is optimized for fluids by a novel loss function that guarantees divergence-free velocity fields at all times. In addition, we demonstrate that we can handle complex parameterizations in reduced spaces, and advance simulations in time by integrating in the latent space with a second network. Our method models a wide variety of fluid behaviors, thus enabling applications such as fast construction of simulations, interpolation of fluids with different parameters, time re-sampling, latent space simulations, and compression of fluid simulation data. Reconstructed velocity fields are generated up to 700× faster than re-simulating the data with the underlying CPU solver, while achieving compression rates of up to 1300×.

- We propose *the first Transport-based Neural Style Transfer (TNST) algorithm for volumetric smoke data.* Our method is able to transfer features from natural images to smoke simulations, enabling general content-aware manipulations ranging from simple patterns to intricate motifs. The proposed algorithm is physically inspired, since it computes the density transport from a source input smoke to a desired target configuration. Our transport-based approach allows direct control over the divergence of the stylization velocity field by optimizing incompressible and irrotational potentials that transport smoke towards stylization. Temporal consistency is ensured by transporting and aligning subsequent stylized velocities, and 3D reconstructions are computed by seamlessly merging stylizations from different camera viewpoints.

- We present a *Neural Style Transfer approach from images to 3D fluids formulated in a Lagrangian viewpoint (LNST)*. Using particles for style transfer has unique benefits compared to grid-based techniques. Attributes are stored on the particles and hence are trivially transported by the particle motion. This intrinsically ensures temporal consistency of the optimized stylized structure and notably improves the resulting quality. Simultaneously, the expensive, recursive alignment of stylization velocity fields of grid approaches is unnecessary, reducing the computation time to less than an hour and rendering neural flow stylization practical in production settings. Moreover, the Lagrangian representation improves artistic control as it allows for multi-fluid stylization and consistent color transfer from images, and the generality of the method enables stylization of smoke and liquids likewise.

- We present the first method to compute a *3D smoke density field directly from 2D artist sketches*, bridging the gap between early stage prototyping of smoke keyframes and visual realization. From the sketch inputs, we compute an initial volume estimate, and refine the density iteratively with a convolutional neural network. Our differentiable sketcher is embedded into the end-to-end training, which results in robust reconstructions. Our training dataset and sketch augmentation strategy is designed such that it enables general applicability. To compute an animated sequence, we interpolate between a pair of reconstructed density keyframes using Wasserstein barycenters. We evaluate the method on synthetic inputs and sketches from artist depicting highly non-physical smoke shapes and motion.

1.4 Publications

This thesis is based on the following peer-reviewed conference publications:

- B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, **Deep Fluids: A Generative Network for Parameterized Fluid Simulations**, *Computer Graphics Forum (Proceedings of Eurographics 2019)*, 38(2), May. 2019.

- B. Kim, V. C. Azevedo, M. Gross, B. Solenthaler, **Transport-Based**

Neural Style Transfer for Smoke Simulations, *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2019)*, 38(6), Nov. 2019.

- B. Kim, V. C. Azevedo, M. Gross, B. Solenthaler, **Lagrangian Neural Style Transfer for Fluids**, *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2020)*, 39(4), Jul. 2020.

This thesis is also based on the following submitted publication:

- X. Huang, G. Cordonnier, B. Kim, V. C. Azevedo, M. Gross, B. Solenthaler, **Deep Reconstruction of 3D Smoke Densities from Artist Sketches**, *submitted to SIGGRAPH Asia 2020*.

This thesis includes the contents of all above papers as well as additional implementation and evaluation details not present in the papers.

During the course of this thesis, the following peer-reviewed papers were published, which are not part of this thesis:

- B. Kim, O. Wang, A. C. Öztireli, M. Gross, **Semantic Segmentation for Line Drawing Vectorization Using Neural Networks**, *Computer Graphics Forum (Proc. Eurographics 2018)*, 37(2), Apr. 2018.
- B. Kim and T. Günther, **Robust Reference Frame Extraction from Unsteady 2D Vector Fields with Convolutional Neural Networks**, *Computer Graphics Forum (Proc. EuroVis 2019)*, 38(3), June. 2019.
- F. Christen, B. Kim, V. C. Azevedo, B. Solenthaler, **Neural Smoke Stylization with Color Transfer**, *Eurographics 2020 Short Paper*, May. 2020.
- S. Biland, V. C. Azevedo, B. Kim, B. Solenthaler, **Frequency-Aware Reconstruction of Fluid Simulations with Generative Networks**, *Eurographics 2020 Short Paper*, May. 2020.
- S. Wiewel, B. Kim, V. C. Azevedo, B. Solenthaler, N. Thuerey, **Latent Space Subdivision: Stable and Controllable Time Predictions for Fluid Flow**, *Computer Graphics Forum (Proc. SCA 2020)*, 39(8), Nov. 2020.
- S. L. Charreyron, Q. Boehler, B. Kim, C. Weibel, C. Chautems, B. J. Nelson, **Modeling Electromagnetic Navigation Systems**, *submitted to IEEE Transaction on Robotics*.

CHAPTER 2

Related Works

We categorize the related works relevant to this thesis in four categories. Firstly, we briefly introduce and review fluid simulations works in Computer Graphics field over the last 30 years distinguishing between Eulerian, Lagrangian, and hybrid discretizations. Then, recent works that apply machine learning for fluids will be discussed, followed by a study of artistically controlling fluid simulations. Lastly, recent deep-learning based stylization approaches are analyzed.

2.1 Fluid Simulation for Computer Graphics

2.1.1 Eulerian Fluids

Incompressible fluids are traditionally simulated by solving the Navier-Stokes (NS) equations [Stokes, 1845], which relate momentum

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f} \tag{2.1}$$

and mass conservation

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

equations, where $\mathbf{u}$ and p are the fluid velocity and pressure with uniform density ρ, and $\mathbf{f}$ represents external forces. The viscosity $\mu\nabla^2\mathbf{u}$ is usually omitted, since simulations for visual effects commonly rely on numerical dissipation instead.

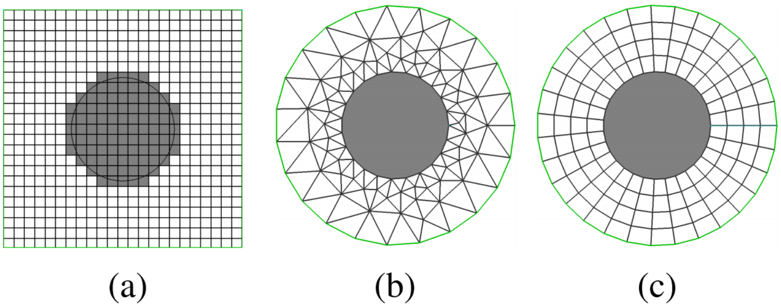


Figure 2.1: *Types of different grids. (a) regular structured grid, (b) unstructured grid, (c) non-regular structured grid. (Image courtesy: [Azevedo and Oliveira, 2013])*

Equations 2.1 and 2.2 are numerically evaluated by discretizing both space and time. One of the approaches for space discretization is to adopt an Eulerian viewpoint: all variables are stored in fixed locations, forming a grid, which can be organized in a structured or unstructured fashion (Figure 2.1). This discretization results in several fields such as density ρ, temperature T, level set ϕ, velocity $\mathbf{u}$, pressure p, vorticity $\boldsymbol{\omega}$, and stream function $\boldsymbol{\Psi}$. Some of these fields are illustrated in Figure 2.2.

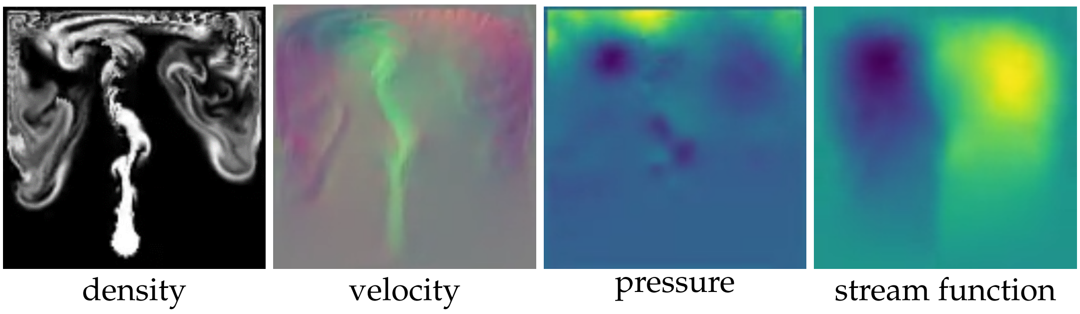


Figure 2.2: *Examples of various 2D fluid field outputs. Velocity is directly mapped to red-green color space after normalization with a fixed blue color value (0.5), and stream function is illustrated by magnitude.*

A common approach to solve the NS equations in Computer Graphics is to employ the fractional step method [Chorin, 1969; Témam, 1969]: the momentum equations are first partially solved for external forces and advection, which is then followed by the computation of a pressure field that enforces incompressibility [Chorin, 1967]. The advection and force integration steps are calculated by

$$\mathbf{u}^* = \mathbf{u} + \Delta t \frac{\partial \mathbf{u}}{\partial t}, \quad \frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \mathbf{f} \tag{2.3}$$

where $\mathbf{u}^*$ is the intermediate velocity field obtained after the integration of external forces and advection.

The first 3D liquid simulation is introduced by [Foster and Metaxas, 1996] with staggered grids [Harlow and Welch, 1965]; however, their explicit integration scheme is inherently unstable with a big time-step and readily caused a performance issue. Thus, Semi-Lagrangian advection with implicit Poisson solver was proposed in [Stam, 1999] for unconditionally stable simulations of fluids. However, it shows severe numerical dissipation due to its first-order integration scheme. Fedkiw et al. [2001] proposed a new model to tackle this shortcoming by applying vorticity confinement [Steinhoff and Underhill, 1994] from the CFD field. Several methods were followed for the same purpose, such as vortex particle [Selle et al., 2005], Back and Forth Error Compensation and Correction (BFECC) [Kim et al., 2005], MacCormack scheme [Selle et al., 2008] and Unsplit Semi-Lagrangian Constrained Interpolation Profile (USCIP) [Kim et al., 2008a].

The divergence-free velocity field $\mathbf{u}'$ (i.e., $\nabla \cdot \mathbf{u}' = 0$) for the next time step is obtained by firstly solving the following Poisson equation with a time step Δt

$$\frac{1}{\rho}\Delta t \nabla^2 p = \nabla \cdot \mathbf{u}^*, \tag{2.4}$$

to get a pressure field p in order to update the intermediate velocity field $\mathbf{u}^*$ by

$$\mathbf{u}' = \mathbf{u}^* - \frac{1}{\rho}\Delta t \nabla p. \tag{2.5}$$

Additionally, the corresponding stream function and vorticity [Ando et al., 2015a] are defined as

$$\begin{aligned} \mathbf{u} &= \nabla \times \boldsymbol{\Psi}, \\ \omega &= \nabla \times \mathbf{u} = \nabla \times (\nabla \times \boldsymbol{\Psi}). \end{aligned} \tag{2.6}$$

Note that $\nabla \cdot (\nabla \times \boldsymbol{\Psi}) = 0$ by construction, and this property will be explored in Section 3.2.3.

In a smoke simulation [Fedkiw et al., 2001], we assume that both density and temperature, two scalar fields, are passively advected by the fluid velocity as

$$\begin{aligned} \frac{\partial \rho}{\partial t} &= -(\mathbf{u} \cdot \nabla)\rho, \\ \frac{\partial T}{\partial t} &= -(\mathbf{u} \cdot \nabla)T. \end{aligned} \tag{2.7}$$

However, both fields reciprocally affect the fluid velocity back in the form of a *buoyancy* force defined as

$$\mathbf{f}_{buoyancy,z} = -\alpha\rho + \beta(T - T_{amb}), \tag{2.8}$$

where α and β are positive weights for density and temperature, respectively, and T_{amp} is ambient temperature.

For liquid simulation, a level set method [Osher and Sethian, 1988] is traditionally used for representing the surface of the liquid. A level set ϕ is an implicit function where the liquid surface is represented as a $\phi(\mathbf{x}) = 0$ iso-contour, and it is during simulation advected as other continuums like density field. Note that the velocity field is usually extrapolated from a level set into the air region for the smoother motion of the liquid surface [Enright et al., 2002] by

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{n} \cdot \nabla \mathbf{u}, \tag{2.9}$$

where $\mathbf{n}$ is a unit normal vector $\nabla\phi/|\nabla\phi|$ orthogonal to the liquid interface. For a broad overview of fluid simulation for computer graphics, we refer to the textbook of Bridson [2015].

2.1.2 Lagrangian Fluids

Complementary to the Eulerian viewpoint, Lagrangian methods discretize fluids equations by particle systems which move with the underlying velocity field. Smoothed Particle Hydrodynamics (SPH) [Lucy, 1977; Gingold and Monaghan, 1977] offer a framework to evaluate continuous functions in discrete particles systems, and they have been employed to a myriad of problems, ranging from astrophysics [Monaghan, 1992], computational fluid dynamics [Monaghan, 1994] to solid mechanics [Libersky and Petschek, 2008]. Since the introduction of SPH to computer graphics [Desbrun and Gascuel, 1996; Müller et al., 2003], various extensions have been presented that made it possible to efficiently simulate millions of particles on a single desktop computer. Accordingly, particle methods reached an unprecedented level of visual quality, where fine-scale surface effects and flow details are reliably captured. To enforce incompressibility, the original state equation based method [Monaghan, 2005; Becker and Teschner, 2007] has been replaced by pressure Poisson equation (PPE) solvers using either a single source term for density invariance [Solenthaler and Pajarola, 2009; Ihmsen et al., 2014] or two PPEs to additionally account for divergence-free velocities [Bender and Koschier, 2015]. Solvers closely related to PPE have been presented, such as Local Poisson SPH [He et al., 2012], Constraint Fluids [Bodin et al., 2012] and Position-based Fluids [Macklin and Müller, 2013].

Boundary handling is computed with particle-based approaches that sample boundary geometry (e.g. [Gissler et al., 2019]) or implicit methods that typically use a signed distance field (e.g. [Koschier and Bender, 2017]). Extensions include highly viscous fluids (e.g. [Peer et al., 2015]), and multiple phases and fluid mixing (e.g. [Ren et al., 2014]). An overview of recent developments in SPH can be found in the course notes of Koschier et al. [2019].

2.1.3 Hybrid Lagrangian-Eulerian Fluids

Hybrid Lagrangian-Eulerian Fluids combine the versatility of the particles representation to track transported quantities with the capacity of grids to enforce incompressibility. Among popular approaches, the Fluid Implicit Particle Method (FLIP) [Brackbill et al., 1988] was first employed in graphics to animate sand and water [Zhu and Bridson, 2005]. Due to its ability to accurately capture sub-grid details it has been widely adopted for liquid simulations, being extended to animation of turbulent water [Kim et al., 2006a], coupled with SPH for modelling small scale splashes [Losasso et al., 2008], improved for efficiency [Ando et al., 2013; Ferstl et al., 2016], used in fluid control [Pan et al., 2013], and enhanced with better particle distribution [Ando and Tsuruno, 2011; Um et al., 2014]. The Material Point Method (MPM) [Stomakhin et al., 2013] was used to simulate a wide class of solid materials [Jiang et al., 2016]. Recent work on hybrid approaches extended the information tracked by the particles by affine [Jiang et al., 2015] and polynomial [Fu et al., 2017] transformations. For a thorough discussion of hybrid continuum models, we refer to Hu et al. [2019c].

2.1.4 Reduced-Order Methods for Fluids

Subspace solvers aim to accelerate simulations by discovering simplified representations. In engineering, these techniques go back decades [Lumley, 1967], but were introduced to fluid simulation in computer graphics by [Treuille et al., 2006] and [Gupta and Narasimhan, 2007]. Since then, improvements have been made to make them controllable [Barbǐ and Popoví, 2008], modular [Wicke et al., 2009], spectral [Long and Reinhard, 2009], consistent with widely-used integrators (e.g., Semi-Lagrangian, MacCormack) [Kim and Delaney, 2013], more energy-preserving [Liu et al., 2015] and memory-efficient [Demby Jones et al., 2016]. A related "Laplacian Eigenfunctions" approach [De Witt et al., 2012] has also been introduced and refined [Gerszewski et al., 2015], removing the need for snapshot training data when computing the linear subspace. Recently, the original Laplacian Eigenfunction method is improved for the scalability in [Cui et al., 2018].

2.2 Machine Learning for Fluids

Combining fluid solvers with machine learning techniques was first demonstrated by [Ladický et al., 2015]. By employing Regression Forests to approximate the Navier-Stokes equations on a Lagrangian system, particle positions and velocities were predicted with respect to input parameters for a next time step. Regression Forests are highly efficient, but require handcrafted features that lack the generality and abstraction power of CNNs.

An LSTM-based method for predicting changes of the pressure field for multiple subsequent time steps has been presented by [Wiewel et al., 2019], resulting in significant speed-ups of the pressure solver. [Morton et al., 2018] also proposed a CNN-based approach for flow dimension reduction but time advancement in an approximation of Koopman subspace. For a single time step, a CNN-based pressure projection was likewise proposed [Tompson et al., 2019; Yang et al., 2016]. These models only replace the pressure projection stage of the solver, and hence are specifically designed to accelerate the enforcement of divergence-freeness.

To visually enhance low resolution simulations, [Chu and Thuerey, 2017] synthesized smoke details by looking up pre-computed patches using CNN-based descriptors, while [Xie et al., 2018] proposed a GAN for super resolution smoke flows in a temporally coherent way. Other works enhance FLIP simulations with a learned splash model [Um et al., 2018], while the deformation learning proposed by [Prantl et al., 2019].

Lattice-Boltzmann steady-state flow solutions are recovered by CNN surrogates using signed distance functions as input boundary conditions in [Guo et al., 2016]. [Farimani et al., 2017] use Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] to train steady state heat conduction and steady incompressible flow solvers. Their method is only demonstrated in 2D and the interpolation capabilities of their architecture are not explored. For both methods, the simulation input is a set of parameterized initial conditions defined over a spatial grid, and the output is a single steady state solution.

Recently, [Umetani and Bickel, 2018] developed a data-driven technique to predict fluid flows around bodies for interactive shape design, while Ma et al. [2018] have demonstrated deep learning based fluid interactions with rigid bodies. Differentiable fluid solvers [Schenck and Fox, 2018;

Hu et al., 2019b; Hu et al., 2019a; Holl et al., 2020] have been introduced that can be automatically coupled with deep learning architectures and provide a natural interface for image-based applications.

2.3 Artistic Control of Fluids

In visual effects production, physics-based simulations are not only used to realistically re-create natural phenomena, but also as a tool to convey stories and trigger emotions. Hence, artistically controlling the shape, motion and the appearance of simulations is essential for providing directability for physics. Specifically to fluids, the major challenge is the non-linearity of the underlying fluid motion equations, which makes optimizations towards a desired target difficult.

2.3.1 Patch-Based Appearance Transfer

Patch-based appearance transfer methods compute similarities between source and target datasets in local neighborhoods, modifying the appearance of the source by transferring best-matched features from the target dataset. Kwatra et al. [2005] employ local similarity measures in an energy-based optimization, enabling texture patches animated by flow fields. This approach was further extended to liquid surfaces [Kwatra et al., 2006; Bargteil et al., 2006], and improved by modifying the texture based on visually salient features of the liquid mesh [Narain et al., 2007].

Jamriška et al. [2015] improved previous work with better temporal coherency and matching precision for obtaining high-quality 2D textured fluids. Texturing liquid simulations was also implemented in a Lagrangian framework by using individually tracked surface patches [Yu et al., 2011; Gagnon et al., 2016; Gagnon et al., 2019].

Image and video-based approaches also take inspiration from fluid transport. Bousseau et al. [2007] proposed a bidirectional advection scheme to reduce patch distortions. Regenerative morphing and image melding techniques were combined with patch-based tracking to produce in-betweens for artist-stylized keyframes [Browning et al., 2014].

Recent advances in patch-based appearance transfer often rely on evaluating the underlying 3D geometric information; examples include improving template matching by a novel similarity measure [Talmi et al., 2017], patch matching for illumination effects [Fišer et al., 2016], extensions to texture mapping [Bi et al., 2017] and intricate texture motifs [Diamanti et al., 2015].

While these approaches were successful in 2D settings and for texturing liquids, they cannot inherently support 3D volumetric data. For a broad discussion of patch-based texture synthesis works we refer to [Barnes and Zhang, 2017].

2.3.2 Velocity Synthesis

Velocity synthesis methods augment flow simulations with velocity fields, which manipulate or enhance volumetric data. Due to the inability of pressure-velocity formulations to properly conserve different energy scales of flow phenomena, sub-grid turbulence [Kim et al., 2008b; Schechter and Bridson, 2008; Narain et al., 2008] was modelled for better energy conservation. These approaches were extended to model turbulence in the wake of solid boundaries [Pfaff et al., 2009], liquid surfaces [Kim et al., 2013] and example-based turbulence synthesis [Sato et al., 2018a].

In order to merge fluids of different simulation instances [Raveendran et al., 2014; Thuerey, 2016] or separated by void regions [Sato et al., 2018a], velocity fields were synthesized by solving an unconstrained energy minimization problem. Ma et al. [2009] synthesized velocity fields with example-based textures for artistic manipulations, but their method is limited to simple 2D patterns.

2.3.3 Fluid Control

Fluid control aims to define the overall shape and behavior through user-specified keyframes or reference images. Optimal [Treuille et al., 2003; McNamara et al., 2004] and proportional-derivative [Fattal and Lischinski, 2004; Shi and Yu, 2005a; Yang et al., 2013] controllers define a set of forces that guide fluid simulation states to desired configurations. These methods were extended to match simulations from different resolutions [Nielsen et al., 2009], geometric potentials [Hong and Kim, 2004] and radial basis functions [Pighin et al., 2004], path-based control [Kim et al., 2006b], guide fluids [Rasmussen et al., 2004; Shi and Yu, 2005b; Thürey et al., 2006; Nielsen and Christensen, 2010; Nielsen et al., 2011], Lagrangian coherent structures [Yuan et al., 2011], volume-preserving morphing [Raveendran et al., 2012], improve performance [Pan and Manocha, 2016], and model more accurate boundary conditions while distinguishing low and high frequencies [Inglis et al., 2017].

However, due to the inherent high dimensionality of the configuration space of fluid solvers, these methods are still computationally challenging, making detailed fluid control hard to achieve. Additionally, they require the specification of target shapes for control, and automatic stylization of fluid features is not possible.

Guided simulation control can also be used to reconstruct target smoke images. Okabe et al. [2015] proposed an appearance transfer method for image-based 3D reconstruction of smoke volumes, while Eckert et al. [2018] utilized proximal operators to reconstruct both the fluid density and motion from single or multiple views.

Optimal controllers were also used to interpolate between distinct simulations [Sato et al., 2018b] and to compute graph-cuts for fluid carving [Flynn et al., 2019]. Feature-based control was applied by Manteaux et al. [2016] for interactive editing and sculpting of pre-computed liquid animations. Space-time features are extracted from surface meshes that enable consistent manipulations. Smoke editing was achieved by deforming the underlying simulation grid [Pan and Manocha, 2017]. Mihalef et al. [2004] presented an approach to control the shape and timing of breaking waves. Vorticity formulations were used to control vortex filaments and rings [Angelidis et al., 2006; Weißmann and Pinkall, 2010]. Smoke simulations were directly modified in the spectral domain [Ren et al., 2013] for appearance control.

2.4 Neural Stylization

2.4.1 Neural Style Transfer

Neural style transfer is the process of rendering image content in different styles by exploring CNNs. The seminal work of Gatys et al. [2016b] was the first to transfer painting styles to natural images. Their model relies on extracting the content of an image by measuring filter responses of a pre-trained CNN, while modelling the style as summary feature statistics. The network's filter responses decompose the image complexity into multiple levels, ranging from low-level features to high-level semantics. Given a target style, NST approaches optimize CNN feature distributions of a source image style, while keeping its original content. Ruder et al. [2018] implemented style transfer for video sequences, addressing temporal coherency issues due to occluded regions and long term correspondences, while Mordvintsev et al. [2018] discuss the impact of different choices of image parameterizations for NST. For a detailed review of NST methods we refer to [Jing et al., 2019].

2.4.2 Differentiable Rendering

Differentiable rendering allows the computation of derivatives of image pixels with respect to the variables used for rendering the image, e.g., vertex positions, normals, colors, camera parameters, etc. These derivatives are crucial to optimization, inverse problems and deep learning backpropagation. Loper and Black [2014] proposed the first raster-based fully differentiable rendering engine with automatically computed derivatives. Anisotropic probing kernels were used to project 3D volumetric data similarly to x-ray scans [Qi et al., 2016]. Tulsiani et al. [2017] used a differentiable ray consistency approach to leverage different types of multi-view observations which can vary from depth and color to foreground masks and normals. Differentiable volume sampling was implemented by Yan et al. [2016] to obtain 2D silhouettes from 3D volumes, adopting a similar sampling strategy as spatial transformer networks [Jaderberg et al., 2015]. Kato et al. [2018] and Liu et al. [2018b] proposed a raster-based differential rendering for meshes with approximate and analytic derivatives, respectively. A cubic stylization algorithm [Liu and Jacobson, 2019] was implemented by minimizing a constrained energy formulation and employed to mesh stylization. Recently, there is a growing interest on differentiable ray marching. Li et al. [2018c] introduces the first general-purpose differentiable ray tracer by removing discontinuities that appear when including visibility terms by directly sampling Dirac delta functions, while a differentiable path-tracer for inverse volumetric rendering with joint estimation of geometry was proposed by Velinov et al. [2018]. [Lombardi et al., 2019] implemented a differentiable ray marcher to take input images to transform into a 3D volume structure with a novel Autoencoder trained in an end-to-end manner. Recently, [Nimier-David et al., 2019] proposed a multi-purposed rendering system - Mitsuba 2 - allowing automatic differentiation for the inverse problem. For an overview on differentiable rendering, we refer to Yifan et al. [2019].

2.4.3 Deep Sketch-Based 3D Reconstruction

We focus our discussion on deep learning based 3D modeling from sketches. For a more general overview of sketch-based systems we refer the interested reader to the course notes of Cordier et al. [2016]. For geometric shape modeling, Convolutional Neural Networks (CNN) were used to build a direct mapping from sketches to parametric 3D shape models [Huang et al., 2017; Nishida et al., 2016], mostly restricted to pre-defined object classes and fixed viewpoints. Delanoy et al. [2018] trained a CNN to predict occupancy in a voxel grid based on a single or multiple contour drawings as input. The method first generates an initial reconstruction with a single-view network, and then uses a so called updater network to iteratively refine the prediction as new drawings from additional views are provided. Li et al. [2018a] uses a CNN to infer the depth and normal maps representing the surface of an object. To reduce ambuigity, the network additionally considers the flow field of the surface to generate a confidence map. The input to the network consist of sketches, silhouette mask, and optional depth sample points and curvature hints. Depth and normal maps were also used to reconstruct a dense point cloud from sketches [Lun et al., 2018]. The decoder captures the object's surface from several viewpoints, which are then fused into a single 3D point cloud through optimization. A deep learning based sketching system has also been used for 3D face and caricature modeling [Han et al., 2017]. 2D lines representing the contours of facial features represent the input to a CNN. An initial sketching mode is followed by sketch and gesture based refinement steps. Deep learning was also used for an interactive modeling tool of 3D hair from 2D sketches [Shen et al., 2020]. Hair contour and a few strokes indicating the hair growing direction are used by a first network to generate a 2D hair orientation field, which is then processed by a second network to output a 3D vector field. For sketch-based garment design, a joint latent shape space is learnt across different modalities representing the draped garment, body shape parameters and garment parameters [Wang et al., 2019]. To our knowledge, no method exists that generates 3D reconstructions of fluids from 2D artist sketches. Sketches have only been used in form of strokes on selected keyframes to control the motion of liquids [Pan et al., 2013]. Local editing is computed with an efficient optimization and then propagated spatially and temporally. Sketched strokes were also used in interactive environments to define shapes and connections of fluid circuits primarily applied in medicine [Zhu et al., 2011] and for 2D flow field design using a generative adversarial network [Hu et al., 2019c].

CHAPTER 3

Deep Generative Model for Fluid Simulations

This chapter is based on the following publication by [Kim et al., 2019b] in collaboration with Technical University of Munich and Pixar:

B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, **Deep Fluids: A Generative Network for Parameterized Fluid Simulations**, *Computer Graphics Forum (Proceedings of Eurographics 2019)*, 38(2), May. 2019.

The supplemental video can be found here: `https://www.youtube.com/watch?v=hSDzOZ9IO8U`.

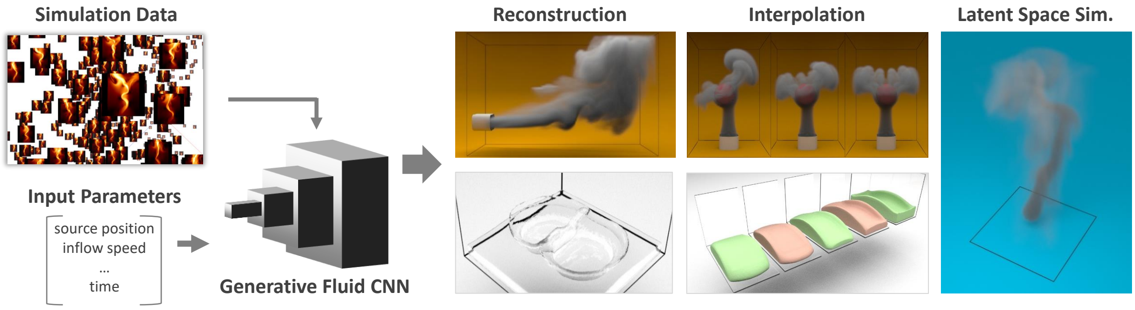


Figure 3.1: *Our generative neural network synthesizes fluid velocities continuously in space and time, using a set of input simulations for training and a few parameters for generation. This enables fast reconstruction of velocities, continuous interpolation and latent space simulations.*

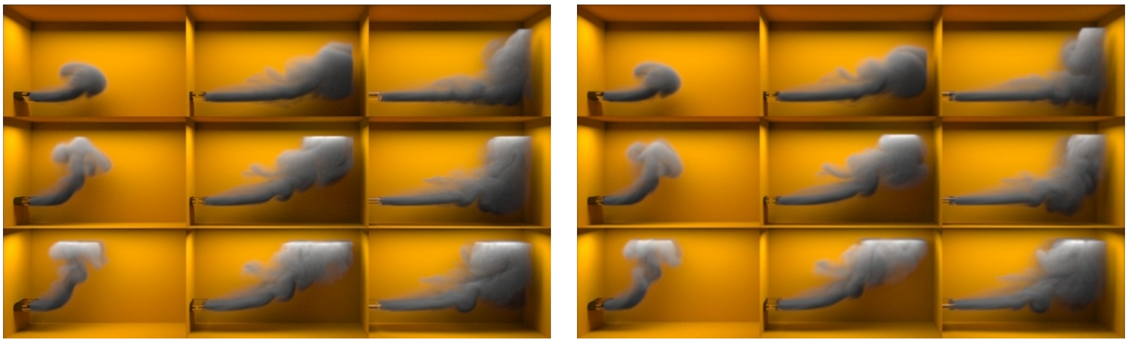


Figure 3.2: *Ground truth (left) and the CNN-reconstructed results (right) for nine sample simulations with varying buoyancy (rows) and inflow velocity (columns). Despite the varying dynamics of the ground truth simulations, our trained model closely reconstructs the reference data.*

3.1 Overview

In this chapter, we propose the first generative neural network that fully constructs dynamic Eulerian fluid simulation velocities from a set of reduced parameters. Given a set of discrete, parameterizable simulation examples, our deep learning architecture generates velocity fields that are incompressible by construction. In contrast to previous subspace methods [Kim and Delaney, 2013], our network achieves a wide variety of fluid behaviors, ranging from turbulent smoke to gooey liquids (Figure 3.1).

The Deep Fluids CNN enhances the state of the art of reduced-order methods (ROMs) in four ways: efficient evaluation time, a natural non-linear representation for interpolation, data compression capability and a novel approach for latent space simulations. Our CNN can generate a full velocity field in constant time, contrasting with previous approaches which are only efficient for sparse reconstructions [Treuille et al., 2006]. Thanks to its 700× speed-ups compared to regular simulations, our approach is particularly suitable for animating physical phenomena in real-time applications such as games, VR and surgery simulators.

Our method is not only capable of accurately and efficiently recovering learned fluid states, but also generates plausible velocity fields for input parameters that have no direct correspondence in the training data. This is possible due to the inherent capability of deep learning architectures to learn representative features of the data. Having a smooth velocity field reconstruction when continuously exploring the parameter space enables various applications that are particularly useful for the prototyping of expensive fluid simulations: fast construction of simulations, interpolation of flu-

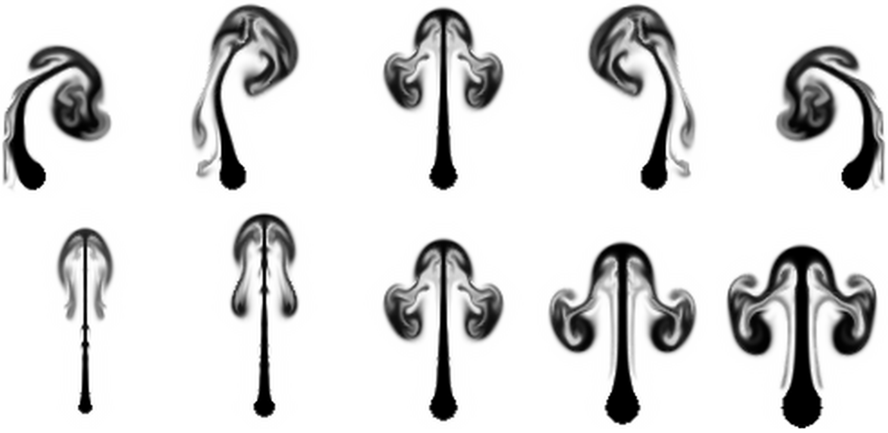


Figure 3.3: *Different snapshots showing the advected densities for varying smoke source parameters. The top and bottom rows show the variation of the initial position source and width, respectively.*

ids with different parameters, and time re-sampling. To handle applications with extended parameterizations such as the moving smoke scene shown in Section 3.4.2, we couple an encoder architecture with a latent space integration network. This allows us to advance a simulation in time by generating a sequence of suitable latent codes.

Additionally, the proposed architecture works as a powerful compression algorithm for velocity fields with compression rates of up to 1300× that outperform previous work [Demby Jones et al., 2016] by two orders of magnitude.

3.2 A Generative Model For Fluids

3.2.1 Challenges

The basis functions used by traditionally reduced-order methods are all linear however, and various methods are then used to coerce the state of the system onto some non-linear manifold. Exploring the use of non-linear functions, as we do here, is a natural evolution. One well-known limitation of reduced-order methods is their inability to simulate liquids because the non-linearity of the liquid interface causes the subspace dimensionality to explode. For example, in solid-fluid coupling, usually the fluid is computed directly while only the solid uses the reduced model [Lu et al., 2016]. Graph-based methods for pre-computing liquid motions [Stanton, 2014] have had some success, but only under severe constraints, e.g., the user viewpoint must be fixed. A reduced-order model with surface-aware basis for liquid simulation is sought in [Ando et al., 2015b]; however, it requires the re-computation of the distance function to surface and modified Poisson problem every step. In contrast, we show that the non-linearity of a CNN-based approach allows it to be applied to liquids without any special design as well as smokes.

3.2.2 CNN Models for Fluids

For a given set of simulated fluid examples, our goal is to train a CNN that approximates the original velocity field dataset. By minimizing loss functions with subsequent convolutions applied to its input, CNNs organize the data manifold into shift-invariant feature maps.

Numerical fluid solvers work by advancing a set of fully specified initial conditions. By focusing on scenes that are initially parameterizable by a handful of variables, such as the position of a smoke source, we are able to generate samples for a chosen class of simulations. Thus, the inputs for our method are parameterizable datasets, and we demonstrate that accurate generative networks can be trained in a supervised way.

3.2.3 Loss Function for Velocity Reconstruction

The network's input is characterized by a pair $[\mathbf{u}_c, \mathbf{c}]$, where $\mathbf{u}_c \in \mathbb{R}^{H \times W \times D \times V_{\text{dim}}}$ is a single velocity vector field frame in V_{dim} dimensions (i.e. $V_{\text{dim}} = 2$ for 2D and $V_{\text{dim}} = 3$ for 3D) with height H, width W and depth D (1 for 2D), generated using the solver's parameters $\mathbf{c} = [c_1, c_2, ..., c_n] \in \mathbb{R}^n$. For the 2D example in Figure 3.3, $\mathbf{c}$ is the combination of x-position and width of the smoke source, and the current time of the frame. Due to the inherent non-linear nature of the Navier-Stokes equations, these three parameters (i.e. position, width, and time) yield a vastly different set of velocity outputs.

For fitting fluid samples, our network uses velocity-parameter pairs and updates its internal weights by minimizing a loss function. This process is repeated by random batches until the network minimizes a loss function over all the training data. While previous works have proposed loss functions for natural images, e.g., L_p norms, MS-SSIM [Zhao et al., 2016], and perceptual losses [Johnson et al., 2016; Ledig et al., 2017], accurate reconstructions of velocity fields have not been investigated. For fluid dynamics, it is especially

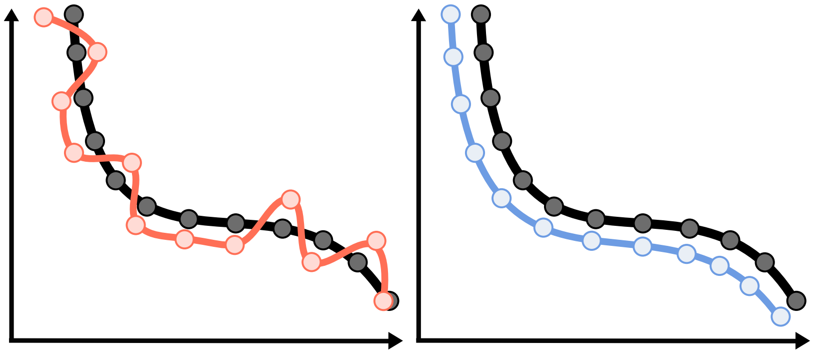


Figure 3.4: *Comparison between reconstructing with L_1 only (left) and L_1 and derivatives (right)*

important to ensure conservation of mass, i.e., to ensure divergence-free motion for incompressible flows. We therefore propose a novel *stream function* based loss function defined as

$$L_G(\mathbf{c}) = ||\mathbf{u_c} - \nabla \times G(\mathbf{c})||_1. \tag{3.1}$$

$G(\mathbf{c}) : \mathbb{R}^n \mapsto \mathbb{R}^{H\times W\times D\times G_{\text{dim}}}$ is the network output and $\mathbf{u_c}$ is a simulation sample from the training data. The curl of the model output is the reconstruction target, and it is *guaranteed* to be divergence-free by construction, as $\nabla \cdot (\nabla \times G(\mathbf{c})) = 0$. Thus, $G(\mathbf{c})$ implicitly learns to approximate a stream function $\boldsymbol{\Psi}_\mathbf{c}$ (i.e. $G_{\text{dim}} = 1$ for 2D and $G_{\text{dim}} = 3$ for 3D) that corresponds to a velocity sample $\mathbf{u_c}$.

While this formulation is highly suitable for incompressible flows, regions with partially divergent motion, such as extrapolated velocities around the free surface of a liquid, are better approximated with a direct velocity inference. For such cases, we remove the curl term from Equation (3.1), and instead use

$$L_G(\mathbf{c}) = ||\mathbf{u_c} - G(\mathbf{c})||_1 \tag{3.2}$$

where the output of G represents a velocity field with $G(\mathbf{c}) : \mathbb{R}^n \mapsto \mathbb{R}^{H\times W\times D\times V_{\text{dim}}}$.

In both cases, simply minimizing the L_1 distance of a high-order function approximation to its original counterpart does not guarantee that their derivatives will match. Consider the example shown in the Figure 3.4: given a function (black line, gray circles), two approximations (red line, left image; blue line, right image) of it with the same average L_1 distances are shown. In the left image, derivatives do not match, producing a jaggy reconstructed behavior; in the right image, both values and derivatives of the L_1 distance are minimized, resulting in matching derivatives. With a sufficiently smooth

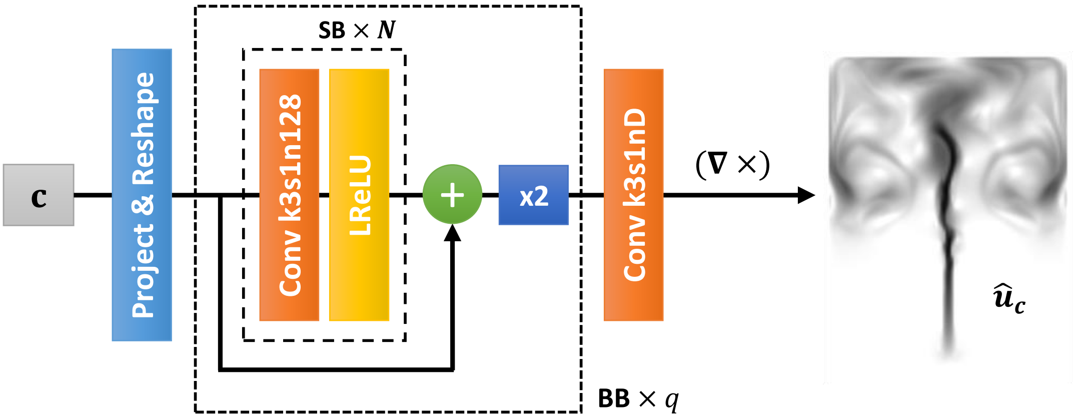


Figure 3.5: *Architecture of the proposed generative model, subdivided into small (SB) and big blocks (BB). Small blocks are composed of flat convolutions followed by a LReLU activation function. Big blocks are composed of sets of small blocks, an additive skip-connection and an upsampling operation. The output of the last layer has D channels (G_{dim} for incompressible velocity fields, V_{dim} otherwise) corresponding to the simulation dimension.*

dataset, high-frequency features of the CNN are in the null space of the L_1 distance minimization and noise may occur as shown in Figure 3.32.

Thus, we augment our loss function to also minimize the difference of the velocity field jacobians. The velocity jacobian $\nabla : \mathbb{R}^{H\times W\times D\times V_{\text{dim}}} \mapsto \mathbb{R}^{H\times W\times D\times (V_{\text{dim}})^2}$ is a second-order tensor that encodes vorticity, shearing and divergence information. Similar techniques, as image gradient difference loss [Mathieu et al., 2016], have been employed for improving frame prediction on video sequences. However, to our knowledge, this is the first architecture to employ jacobian information to improve velocity field data. Our resulting loss function is defined as

$$L_G(\mathbf{c}) = \lambda_{\mathbf{u}}||\mathbf{u_c} - \hat{\mathbf{u}}_{\mathbf{c}}||_1 + \lambda_{\nabla\mathbf{u}}||\nabla\mathbf{u_c} - \nabla\hat{\mathbf{u}}_{\mathbf{c}}||_1, \tag{3.3}$$

where $\hat{\mathbf{u}}_{\mathbf{c}} = \nabla \times G(\mathbf{c})$ for incompressible flows and $\hat{\mathbf{u}}_{\mathbf{c}} = G(\mathbf{c})$ for compressible flows, and $\lambda_{\mathbf{u}}$ and $\lambda_{\nabla\mathbf{u}}$ are weights used to emphasize the reconstruction of either the velocities or their derivatives. In practice, we used $\lambda_{\mathbf{u}} = \lambda_{\nabla\mathbf{u}} = 1$ for normalized data (see Section 3.5.1). The curl of the velocity and its jacobians are computed internally by our architecture and do not need to be explicitly included in the dataset.

3.2.4 Implementation

For the implementation of our generative model we adopted and modified the network architecture from [Li et al., 2018d]. As illustrated in Figure 3.5, our generator starts by projecting the initial $\mathbf{c}$ parameters into an m-dimensional vector of weights $\mathbf{m}$ via fully connected layers. The dimension of $\mathbf{m}$ depends on the network output $\mathbf{d} = [H, W, D, V_{\text{dim}}]$ and on a custom defined parameter q. With $d_{\text{max}} = \max(H, W, D)$, q is calculated by $q = \log_2(d_{\text{max}}) - 3$, meaning that the minimum supported number of cells in one dimension is 8. Additionally, we constrain all our grid dimensions to be divisible by 2^q. Since we use a fixed number of feature maps per layer, the number of dimensions of $\mathbf{m}$ is $m = \frac{H}{2^q} \times \frac{W}{2^q} \times \frac{D}{2^q} \times 128$ and those will be expanded to match the original output resolution.

The m-dimensional vector $\mathbf{m}$ is then reshaped to a $[\frac{H}{2^q}, \frac{W}{2^q}, \frac{D}{2^q}, 128]$ tensor. As shown in Figure 3.5, the generator component is subdivided into small (SB) and big blocks (BB). For small blocks, we perform N (most of our examples used $N = 4$ or 5) flat convolutions followed by Leaky Rectified Linear Unit (LReLU) activation functions [Maas et al., 2013].

We substituted the Exponential Liner Unit (ELU) activation function in the original method from [Li et al., 2018d] by the LReLU as it yielded sharper outputs when minimizing the L_1 loss function. Additionally, we employ residual skip connections [He et al., 2016], which are an element-wise sum of feature maps of input and output from each SB. While the concatenative skip connections employed by [Li et al., 2018d] are performed between the first hidden states and the consecutive maps with doubling of the depth size to 256, ours are applied to all levels of SBs with a fixed size of 128 feature maps. After the following upsample operation, the dimension of the output from a BB after i passes is $[\frac{H}{2^{q-i}}, \frac{W}{2^{q-i}}, \frac{D}{2^{q-i}}, 128]$. Our experiments showed that performing these reductions to the feature map sizes with the residual concatenation improved the network training time without degradation of the final result.

3.3 Extended Parameterizations

Scenes with a large number of parameters can be challenging to parameterize. For instance, the dynamically moving smoke source example (Figure 3.17) can be parameterized by the history of control inputs, i.e., $[\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_t] \rightarrow \mathbf{u}_t$, where $\mathbf{p}_t$ and $\mathbf{u}_t$ represent the smoke source position and the reconstructed velocity field at time t, respectively. In this case, however, the number of parameters grows linearly with the number of frames tracking user inputs. As a consequence, the parameter space would be infeasibly large for data-driven approaches, and would be extremely costly to cover with samples for generating training data.

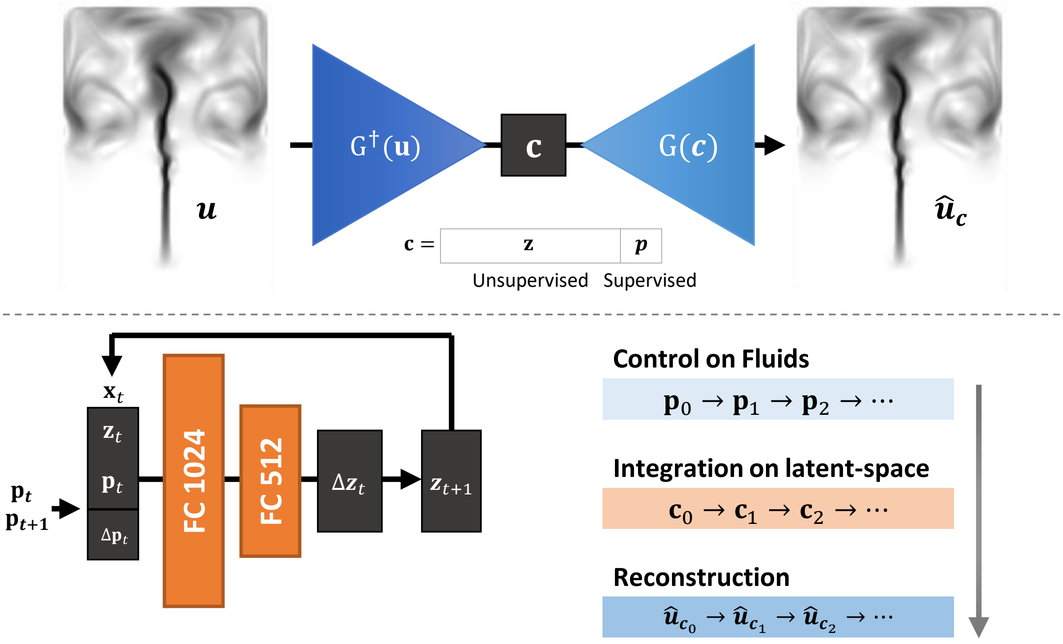


Figure 3.6: *Autoencoder (top) and latent space integration network (bottom). The autoencoder compresses a velocity field* $\mathbf{u}$ *into a latent space representation* $\mathbf{c}$*, which includes a supervised and unsupervised part (*$\mathbf{p}$ *and* $\mathbf{z}$*). The latent space integration network finds mappings from subsequent latent code representations* $\mathbf{c}_t$ *and* $\mathbf{c}_{t+1}$*.*

To extend our approach to these challenging scenarios, we add an encoder architecture $G^{\dagger}(\mathbf{u}) : \mathbb{R}^{H\times W\times D\times V_{\text{dim}}} \mapsto \mathbb{R}^n$ to our generator of Section 3.2, and combine it with a second smaller network for time integration (Section 3.3.1), as illustrated in Figure 3.6. In contrast to our generative network, the encoder architecture maps velocity field frames into a parameterization $\mathbf{c} = [\mathbf{z}, \mathbf{p}] \in \mathbb{R}^n$, in which $\mathbf{z} \in \mathbb{R}^{n-k}$ is a reduced latent space that models arbitrary features of the flow in an unsupervised way and $\mathbf{p} \in \mathbb{R}^k$ is a supervised parameterization to control specific attributes [Kulkarni et al., 2015]. Note that this separation makes the latent space sparser while training, which in turn improves the quality of the reconstruction. For the moving smoke source example in Section 3.4.2, $n = 16$ and $\mathbf{p}$ encodes x, z positions used to control the position of the smoke source.

The combined encoder and generative networks are similar to Deep Convolutional autoencoders [Vincent et al., 2010], where the generative network $G(\mathbf{c})$ acts as a decoder. The encoding architecture is symmetric to our generative model, except that we do not employ the inverse of the curl operator and the last convolutional layer. We train both generative and encoding networks with a combined loss similar to Equation (3.3), as

$$L_{AE}(\mathbf{u}) = \lambda_{\mathbf{u}}||\mathbf{u_c} - \hat{\mathbf{u}}_\mathbf{c}||_1 + \lambda_{\nabla\mathbf{u}}||\nabla\mathbf{u_c} - \nabla\hat{\mathbf{u}}_\mathbf{c}||_1 + \lambda_{\mathbf{p}}||\mathbf{p} - \hat{\mathbf{p}}||_2^2, \qquad (3.4)$$

where $\hat{\mathbf{p}}$ is the part of the latent space vector constrained to represent control parameters $\mathbf{p}$, and $\lambda_\mathbf{p}$ is a weight to emphasize the learning of supervised parameters. Note that the L_2 distance is applied to control parameters unlike vector field outputs, as it is a standard cost function in linear regression. As before, we used $\lambda_\mathbf{u} = \lambda_{\nabla\mathbf{u}} = \lambda_\mathbf{p} = 1$ for all our normalized examples (Section 3.5.1). With this approach we can handle complex parameterizations, since the velocity field states are represented by the remaining latent space dimensions in $\mathbf{z}$. This allows us to use latent spaces which do not explicitly encode the time dimension as a parameter. Instead, we can use a second latent space integration network that generates a suitable sequence of latent codes.

3.3.1 Latent Space Integration Network

The latent space only learns a diffuse representation of time by the velocity field states $\mathbf{z}$. Thus we propose a latent space integration network for advancing time from reduced representations. The network $T(\mathbf{x}_t) : \mathbb{R}^{n+k} \mapsto \mathbb{R}^{n-k}$ takes an input vector $\mathbf{x}_t = [\mathbf{c}_t; \Delta\mathbf{p}_t] \in \mathbb{R}^{n+k}$ which is a concatenation of a latent code $\mathbf{c}_t$ at current time t and a control vector difference between user input parameters $\Delta\mathbf{p}_t = \mathbf{p}_{t+1} - \mathbf{p}_t \in \mathbb{R}^k$. The parameter $\Delta\mathbf{p}_t$ has the same dimensionality k as the supervised part of our latent space, and serves as a transition guidance from latent code $\mathbf{c}_t$ to $\mathbf{c}_{t+1}$. The output of $T(\mathbf{x}_t)$ is the residual $\Delta\mathbf{z}_t$ between two consecutive states. Thus, a new latent code is computed with $\mathbf{z}_{t+1} = \mathbf{z}_t + T(\mathbf{x}_t)$ as seen in Figure 3.6.

For improved accuracy we let T look ahead in time, by training the network on a window of w sequential latent codes with an L_2 loss function:

$$L_T(\mathbf{x}_t, ..., \mathbf{x}_{t+w-1}) = \frac{1}{w}\sum_{i=t}^{t+w-1} ||\Delta\mathbf{z}_i - T_i||_2^2, \qquad (3.5)$$

where T_i is recursively computed from t to i. Our window loss Equation (3.5) is designed to minimize not only errors on the next single step integration but also errors accumulated in repeated latent space updates. We found that $w = 30$ yields good results, and a discussion of the effects of different values of w is provided in the Section 3.5.3.

We realize T as a multilayer perceptron (MLP) network. The rationale behind choosing MLP instead of LSTM is that T is designed to be a navigator

Algorithm 1: *Simulation with the Latent Space Integration Network*

```
$\mathbf{c}_0 \leftarrow G^\dagger(\mathbf{u}_0)$
while simulating from $t$ to $t+1$ do
    $\mathbf{x}_t \leftarrow [\mathbf{c}_t; \Delta\mathbf{p}_t]$   // $\mathbf{c}_t$ from previous step, $\mathbf{p}_{t+1}$ is given
    $\mathbf{z}_{t+1} \leftarrow \mathbf{z}_t + T(\mathbf{x}_t)$   // latent code inference
    $\mathbf{c}_{t+1} \leftarrow [\mathbf{z}_{t+1}; \mathbf{p}_{t+1}]$
    $\mathbf{u}_{t+1} \leftarrow G(\mathbf{c}_{t+1})$   // velocity field reconstruction
end while
```

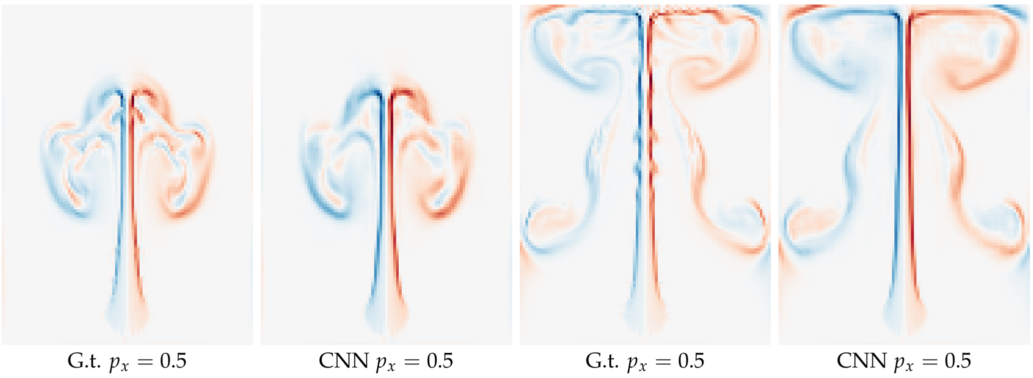


Figure 3.7: *Vorticity plot of a 2D smoke simulation with direct correspondences to the training dataset for two different times. The RdBu colormap is used to show both the magnitude and the rotation direction of the vorticity (red: clockwise). Our CNN is able to closely approximate ground truth samples (G.t.).*

on the manifold of the latent space, and we consider these integrations as controlled individual steps rather than physically induced ones. The network consists of three fully connected layers coupled with ELU activation functions. We employ batch normalization and dropout layers with probability of 0.1 to avoid overfitting.

Once the networks $G, G^\dagger$ and T are trained, we use Algorithm (1) to reconstruct the velocity field for a new simulation. The algorithm starts from an initial reduced space that can be computed from an initial incompressible velocity field. The main loop consists of concatenating the reduced space and the position update into $\mathbf{x}_t$; then the latent space integration network computes $\Delta\mathbf{z}_t$, which is used to update $\mathbf{c}_t$ to $\mathbf{c}_{t+1}$. Finally, the generative network G reconstructs the velocity field $\mathbf{u}_{t+1}$ by evaluating $\mathbf{c}_{t+1}$.

3.4 Results

In the following we demonstrate that our Deep Fluids CNN can reliably recover and synthesize dynamic flow fields for both smoke and liquids. We refer the reader to the supplemental video for the corresponding animations. For each scene, we reconstruct velocity fields computed by the generative network and advect densities for smoke simulations or surfaces for liquids. Vorticity confinement or turbulence synthesis were not applied after the network's reconstruction, but such methods could be added as a post-processing step. We trained our networks using the Adam optimizer [Kingma and Ba, 2015] for 300,000 iterations with varying batch sizes to maximize GPU memory usage (8 for 2D and 1 for 3D). For the time network T, we use 30,000 iterations. The learning rate of all networks is scheduled by a cosine annealing decay [Loshchilov and Hutter, 2017], where we use the learning range from [Smith, 2017]. Scene settings, computation times and memory consumptions are summarized in Table (3.1). Fluid scenes were computed with *mantaflow* [Thuerey and Pfaff, 2018] using an Intel i7-6700K CPU at 4.00 GHz with 32GB memory, and CNN timings were evaluated on a 8GB NVIDIA GeForce GTX 1080 GPU. Networks are trained on a 12GB NVIDIA Titan X GPU.

3.4.1 2D Smoke Plume

A sequence of examples that portray varying, rising smoke plumes in a rectangular container is shown in Figure 3.3, where advected densities for different initial source positions (top) and widths (bottom) are shown. Since visualizing the advected smoke may result in blurred flow structures, we display vorticities instead, facilitating the understanding of how our CNN is able to reconstruct and interpolate between samples present in the dataset. Additionally, we use the *hat* notation to better differentiate parameters that do not have a direct correspondence with the ground truth data (e.g., $\hat{\mathbf{p}}_x$ for an interpolated position on the x-axis). Our training set for this example consists of the combination of 5 samples with varying source widths w and 21 samples with varying x positions $\mathbf{p}_x$. Each simulation is computed for 200 frames, using a grid resolution of 96×128 and a domain size of $(1, 1.\overline{33})$. The network is trained with a total of $21,000$ unique velocity field samples.

Reconstruction with Direct Correspondences to the Dataset To analyze the reconstruction power of our approach, we compare generated velocities for parameters which have a direct correspondence to the original

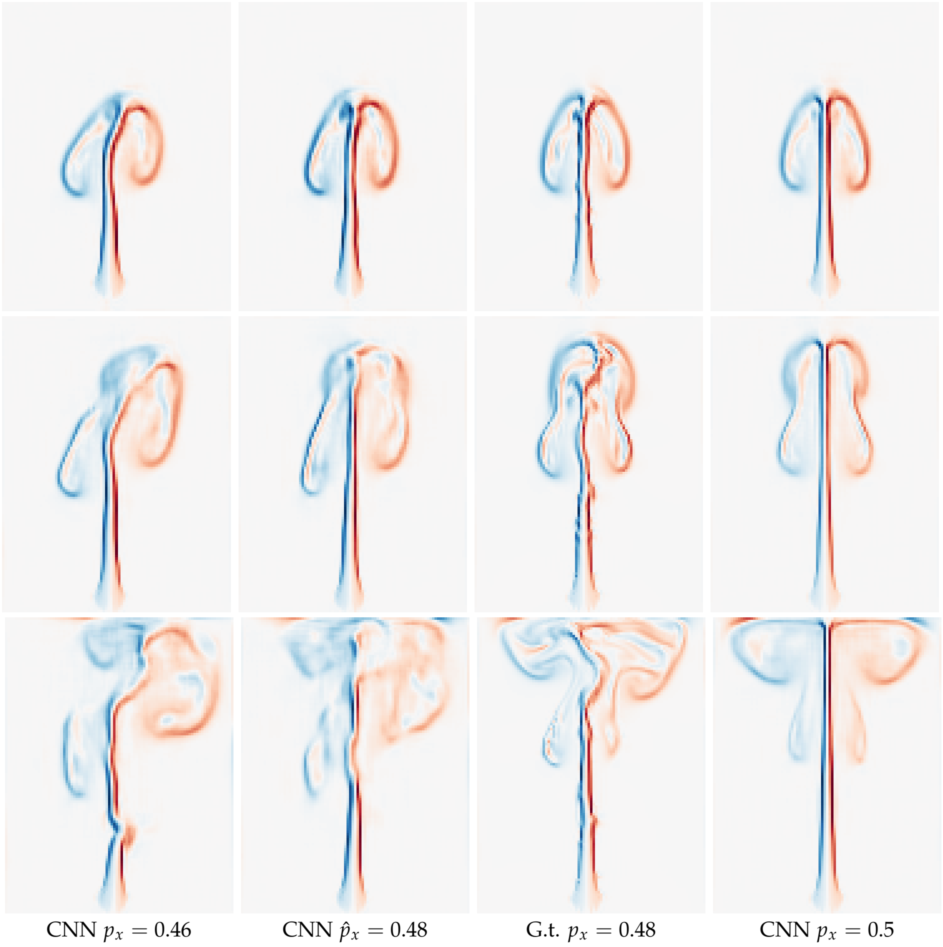


Figure 3.8: *Vorticity plot of a 2D smoke simulation showing CNN reconstructions at ground truth correlated positions* $\mathbf{p}_x = 0.46$ *and* $\mathbf{p}_x = 0.5$*, the interpolated result at* $\hat{\mathbf{p}}_x = 0.48$*, and ground truth (G.t.) at* $\hat{\mathbf{p}}_x = 0.48$ *which is not part of the training dataset.*

dataset, i.e. the ground truth samples. Figure 3.7 shows vorticity plots comparing the ground truth (G.t.) and our CNN output for two different frames. The CNN shows a high reconstruction quality, where coarse structures are almost identically reproduced, and fine structures are closely approximated.

Sampling at Interpolated Parameters We show the interpolation capability of our approach in Figure 3.8. Left and right columns show the CNN reconstructions at ground truth correlated positions $\mathbf{p}_x = 0.46$ and $\mathbf{p}_x = 0.5$, while the second column shows a vorticity plot interpolated at $\hat{\mathbf{p}}_x = 0.48$. The third column shows the simulated ground truth for the same position. For positions not present in the original data, our CNN synthesizes plausible new motions that are close to ground truth simulations. Figure 3.9

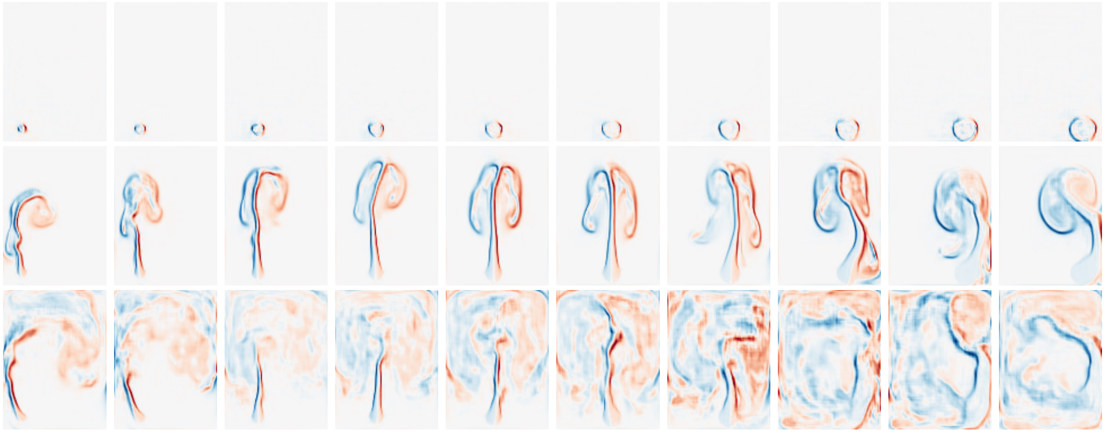


Figure 3.9: *Several vorticity plots for the 2D smoke CNN reconstruction. Each row shows the variation of both position and size of the initial smoke source. Top row corresponds to the first time frame, while the middle and bottom rows show frame* 100 *and* 200*, respectively.*

shows vorticity plots of our CNN reconstructions for varying both smoke source position and width - smaller left located plumes are shown on the left, while bigger right located plumes are shown on the right. The top row shows the first frame of the simulation, while the middle and bottom rows show frames 100 and 200, respectively.

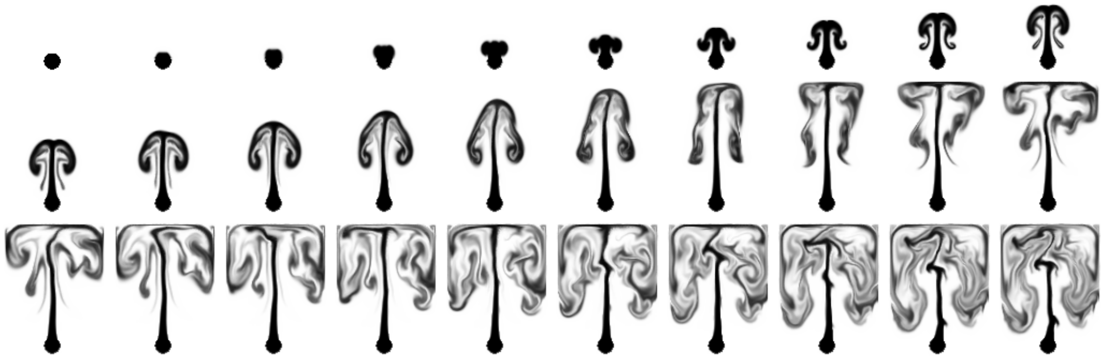


Figure 3.10: *Different snapshots in time where density is advected by the reconstructed velocity fields at interpolated position* $\hat{p}_x = 0.47$*. All outputs represent realistic fluid flows.*

Advected Density using Velocity Reconstruction vs. Direct Density Reconstruction In Figure 3.10, we show different snapshots for the time evolution of density values at an interpolated smoke source position $\hat{p}_x = 0.47$,

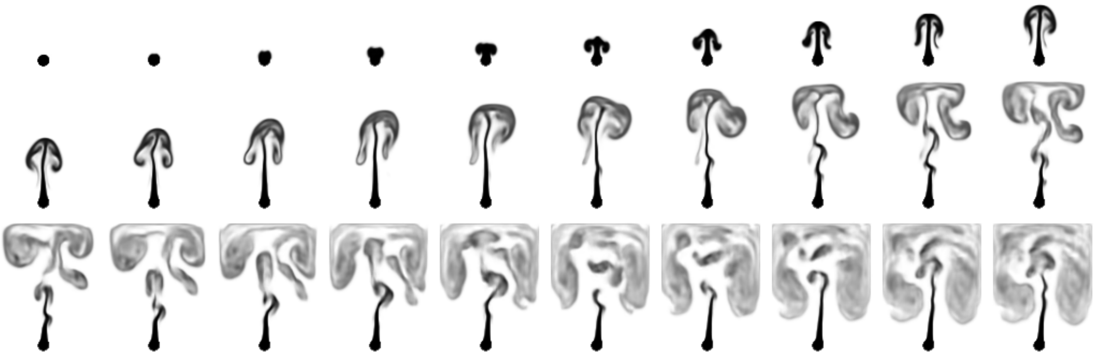


Figure 3.11: *Different snapshots in time for a network trained* directly *on a density-only dataset. The smoke motion is less lively and towards the end of the sequence the main smoke stream breaks apart.*

with no correspondences on the trained dataset. Finally, Figure 3.11 shows the result of training a network directly using density values instead of velocity fields. The motion of densities is less lively, and the smoke stream breaks up for later frames, yielding unrealistic simulation outputs.

3.4.2 3D Smoke Examples

Smoke & Sphere Obstacle Figure 3.12 shows a 3D example of a smoke plume interacting with a sphere computed on a grid of size 64 × 96 × 64. The training data consists of ten simulations with varying sphere positions, with the spaces between spheres centroid samples consisting of 0.06 in the interval $[0.2, 0.8]$. The left and right columns of Figure 3.12 show the CNN-reconstructed simulations at positions $\mathbf{p}_x = 0.44$ and $\mathbf{p}_x = 0.5$, while the second column presents the interpolated results using our generative network at $\hat{\mathbf{p}}_x = 0.47$. Even with a sparse and hence challenging training dataset, flow structures are plausibly reconstructed and compare favorably with ground truth simulations (third column) that were not present in the original dataset.

Smoke Inflow and Buoyancy A collection of simulations with varying inflow speed (columns) and buoyancy (rows) is shown in Figure 3.2 for the ground truth (left) and our generative network (right). We generated 5 inflow velocities (in the range $[1.0, 5.0]$) along with 3 different buoyancy values (from 6×10^{-4} to 1×10^{-3}) for 250 frames. Thus, the network was trained with $3,750$ unique velocity fields. Figure 3.13 demonstrates an interpolation

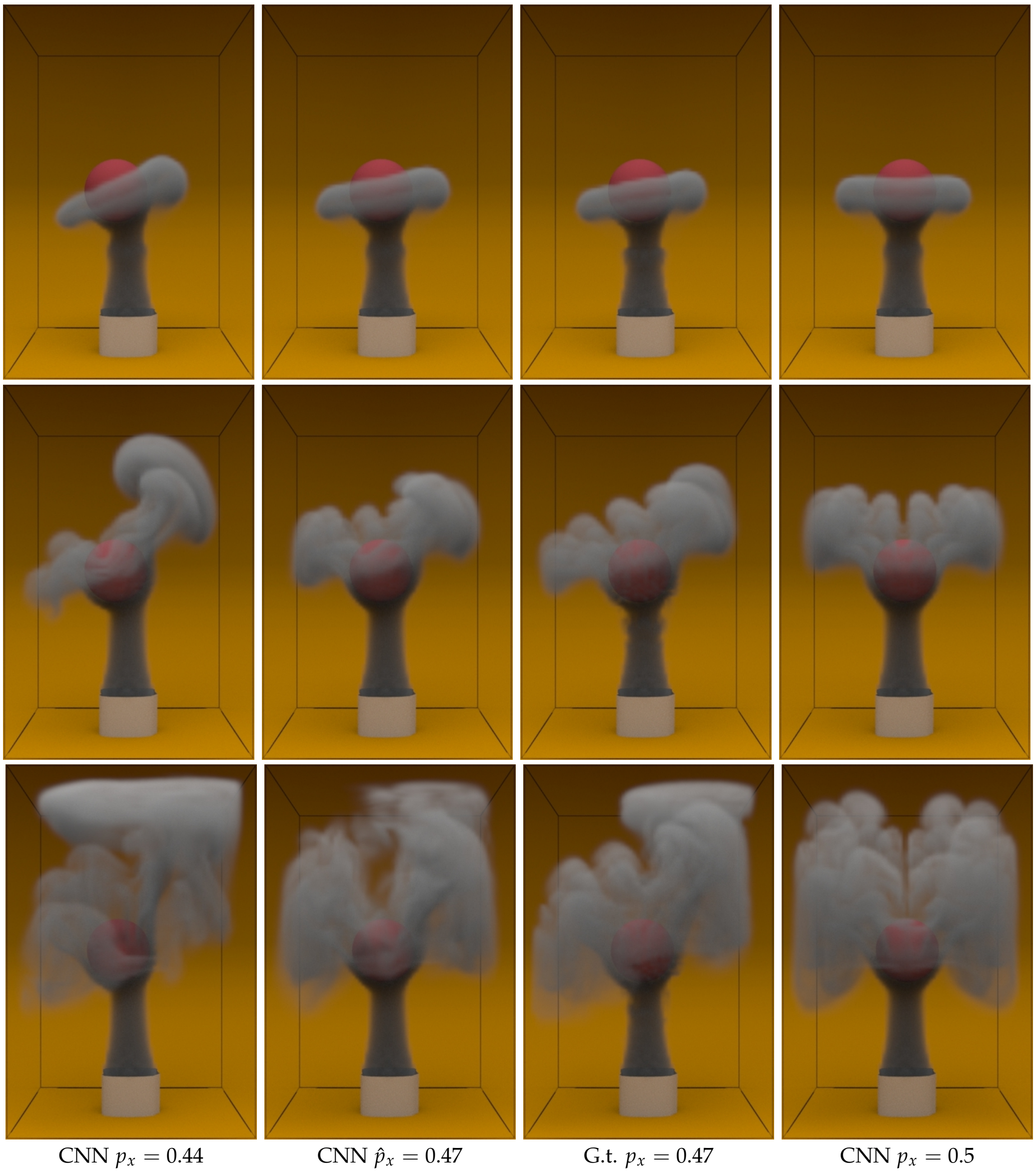


Figure 3.12: *Interpolated result (second column) given two input simulations (left and right) with different obstacle positions on the x-axis. Our method results in plausible in-betweens compared to ground truth (third column) even for large differences in the input.*

example for the buoyancy parameter. The generated simulations on the left and right (using a buoyancy of 6×10^{-4} and 1×10^{-3}) closely correspond to the original dataset samples, while the second simulation is reconstructed by our CNN using an interpolated buoyancy of 8×10^{-4}. We show the ground truth simulation on the third image for a reference comparison. Our method recovers structures accurately, and the plume shape matches the reference ground truth.

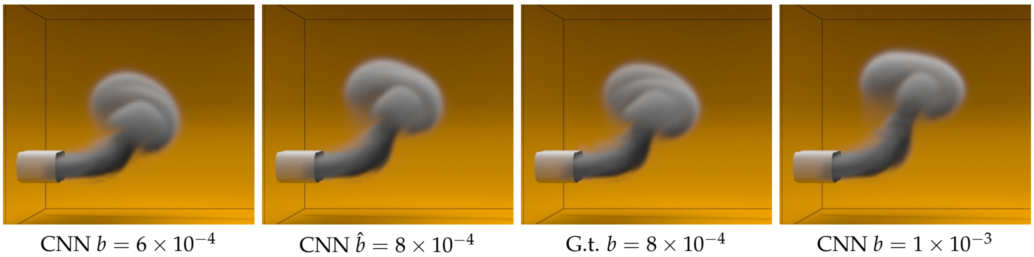

CNN $b = 6 \times 10^{-4}$ CNN $\hat{b} = 8 \times 10^{-4}$ G.t. $b = 8 \times 10^{-4}$ CNN $b = 1 \times 10^{-3}$

Figure 3.13: *Reconstructions of the rising plume scene (left and right), reconstruction for an interpolated buoyancy value ($\hat{b} = 8 \times 10^{-4}$) (second image) and the corresponding ground truth (third image).*

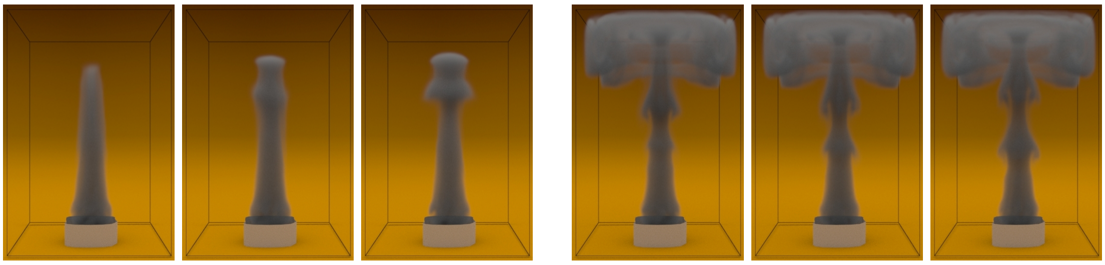


Figure 3.14: *Trained with five different grid resolutions, our network is able interpolate resolutions of* $40 \times 60 \times 40$ *(middle column, left image) and* $88 \times 132 \times 88$ *(middle column, right image)*

Resolution Interpolation Figure 3.14 shows the capability of our approach to interpolate between unusual input parameters such as the grid resolution. We simulate a rising smoke plume in 3D with five different grid sizes: $32 \times 48 \times 32$, $48 \times 64 \times 48$, $64 \times 96 \times 64$, $80 \times 120 \times 80$ and $96 \times 144 \times 96$. We then upsample the velocities of the coarser grid resolutions with bicubic interpolation to the finest one, and feed these to our network. Grid resolutions are uniformly scaled, and we are able to map these to a single network input that varies from 0 to 1. We reduced the feature maps to 64 (half of the other examples) as otherwise the highest grid resolution was too large to train with our current hardware. Results of interpolations between different simulation resolutions can be seen in Figure 3.14.

Rotating Smoke We trained our autoencoder and latent space integration network for a smoke simulation with a periodically rotating source using 500 frames as training data. The source rotates in the XZ-plane with a period of 100 frames. This example is designed as a stress test for extrapolating time using our latent space integration network. In Figure 3.15, we show that our approach is able to correctly capture the periodicity present in the original dataset. Moreover, the method successfully generates another 500 frames, resulting in a simulation that is 100% longer than the original data.

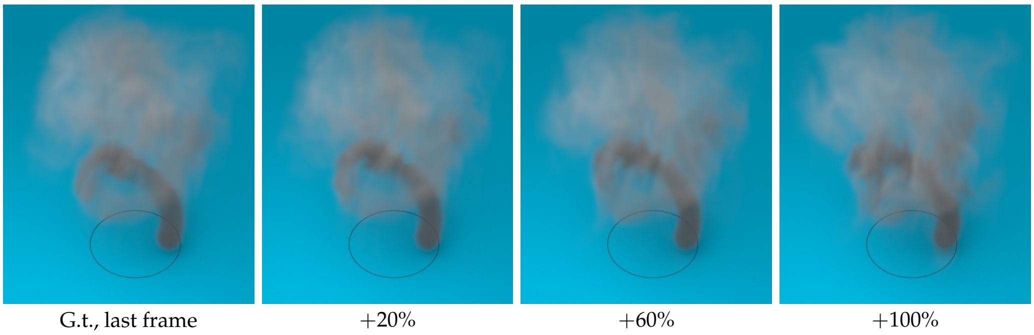


Figure 3.15: *Time extrapolation results using our latent space integration network. The left image shows the last frame of the ground truth simulation. The subsequent images show results with time extrapolation of +20%, +60% and +100% of the original frames.*

Moving Smoke A smoke source is moved in the XZ-plane along a path randomly generated using Perlin noise [Perlin, 1985]. We sampled 200 simulations on a grid of size $48 \times 72 \times 48$ for 400 frames - a subset is shown in Figure 3.16 - and used them to train our autoencoder and latent space integration networks. In Figure 3.17, we show a moving smoke source whose motion is not part of the training data and was computed by integrating in the latent space. We extrapolate in time to increase the simulation duration by 100% (i.e., 800 frames). The network generates a plausible flow for this unseen motion over the full course of the inferred simulation. Although the results shown here were rendered offline, the high performance of our trained model would allow for interactive simulations.

3.4.3 2D Liquid Drop

Additional results for a 2D liquid drop are shown as velocity magnitude plots in Figure 3.18, Figure 3.19 and Figure 3.20. Figure 3.18 shows various reconstructions of varying both initial drop size and x-position. Note that we use non-extrapolated velocity fields for training and reconstruction to see how accurate the reconstruction is around interfaces. We use intervals of $[0.2, 0.8]$ and $[0.04, 0.08]$ with 10 samples for the position and 4 samples for size, respectively. In Figure 3.19, we compare our results side-by-side against ground truth simulations, where the top and bottom columns show time frame 37 and 60 respectively. The first, third and fifth columns show

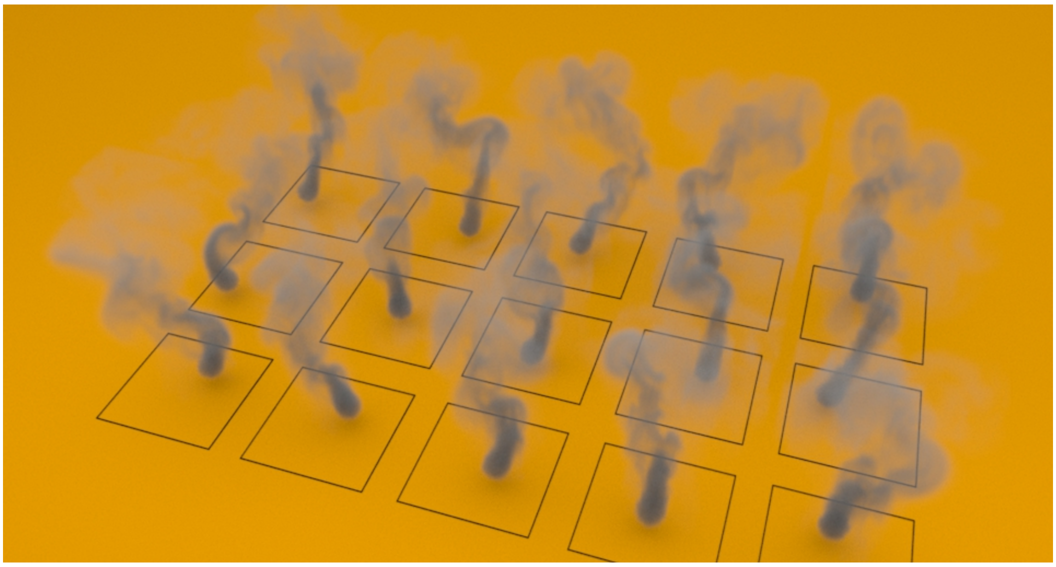


Figure 3.16: *Example simulations of the moving smoke scene used for training the extended parameterization networks.*

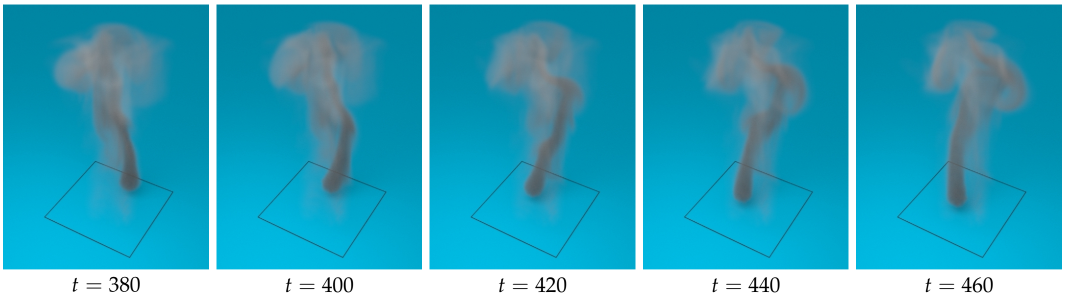


Figure 3.17: *Different snapshots of a moving smoke source example simulated in the latent space.*

ground truth simulations while the second, fourth and sixth columns show the reconstructed counterparts. We notice small differences on splash formation, but the overall bulk of the liquid is respected.

For all data-driven approaches the quality of a trained model, and hence in our case the quality of the simulation results, depend on the training data that is used. One important factor is the sampling density that is used for the training. Especially for generating in-betweens, one has to analyze how dense input simulations must be sampled, in order to avoid artifacts in the reconstruction.

We have conducted a test with a liquid drop scene that has different x-positions in the training simulations. In our test, we varied the number of initial liquid drops - and hence the number of simulations in the training

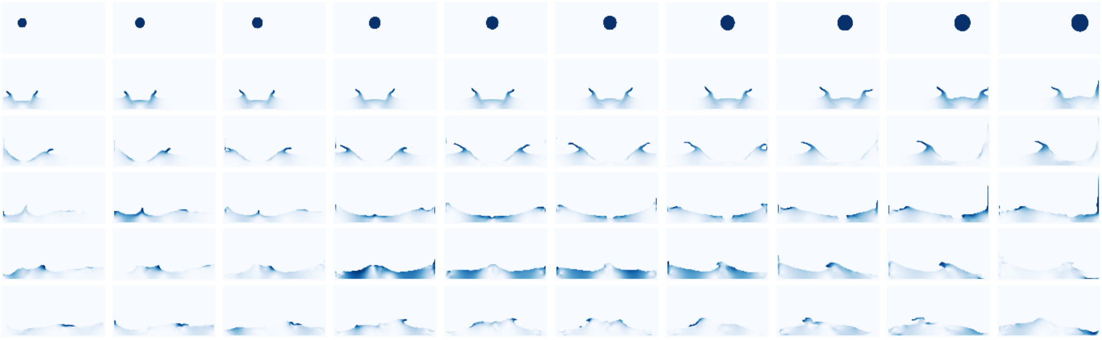


Figure 3.18: *Velocity magnitude plots for our CNN reconstruction on a 2D liquid drop scenario. Each column shows the variation of both position and size of the initial liquid drop. The following rows show time frames* 37, 70, 100, 150, 200, *respectively.*

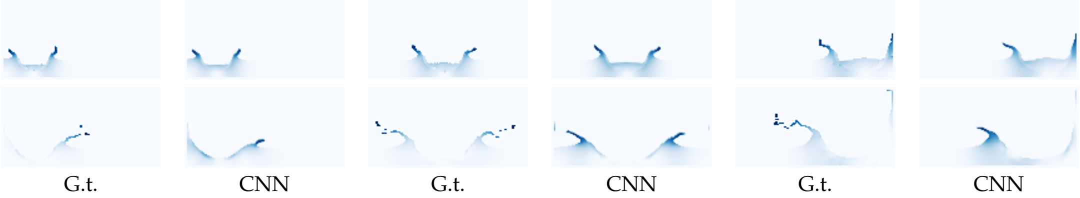


Figure 3.19: *Comparisons between* ground truth *and reconstructed liquid data. Odd columns (first, third and fifth) show ground truth data, while even columns (second, fourth and sixth) show the reconstruction of the CNN.*

phase - to evaluate how this impacts the ability of the network to generate interpolated results. In particular, we want to understand when the network fails to reconstruct proper liquid interpolations. Figure 3.20 shows three different sampling densities with gray circles indicating the initial liquid drop in the training dataset: 14 (left), 12 (middle) and 10 input samples (right). Different networks were trained with the corresponding number of samples, and intermediate liquid drop positions are generated with the trained models between the first pair of discrete samples for each scene, indicated by the red box.

The second row shows the reconstruction results where the accuracy drops as the sampling density decreases. Hence, we argue that some overlap between noticeable scene features is necessary in order to avoid reconstruction artifacts. Automatically quantifying how good an initial training dataset is regarding interpolation is left as future work.

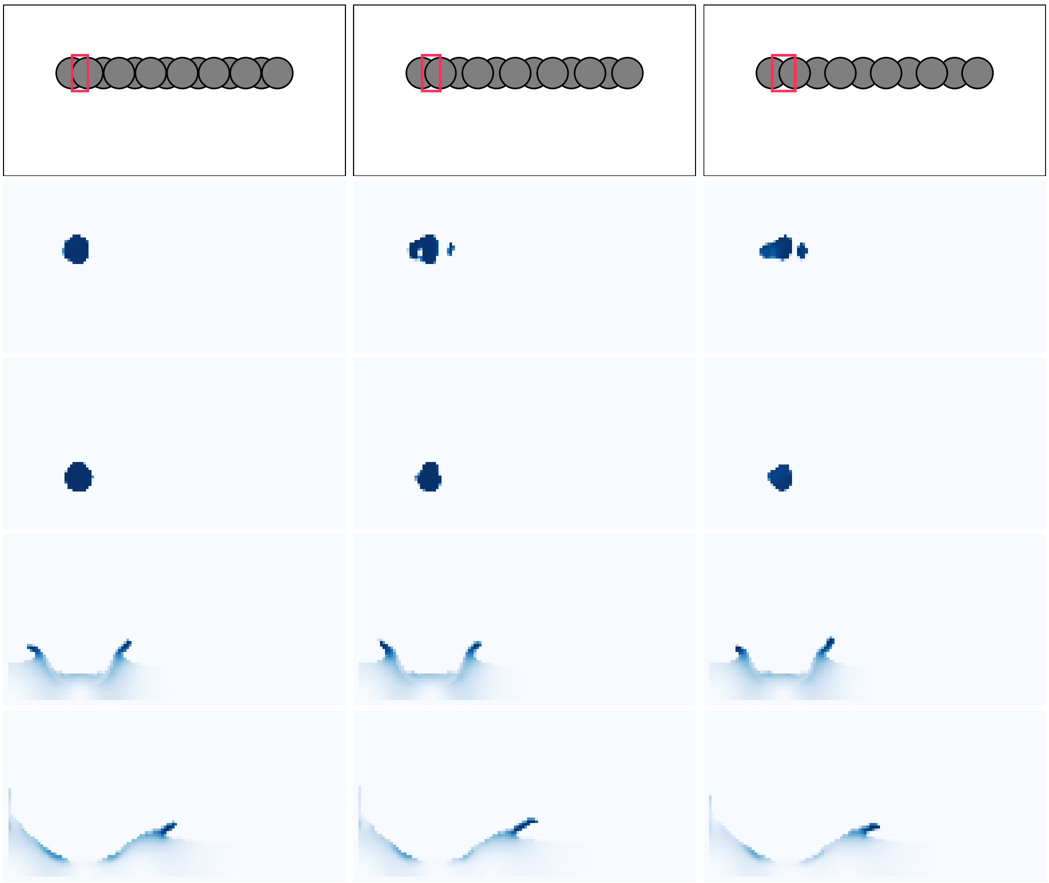
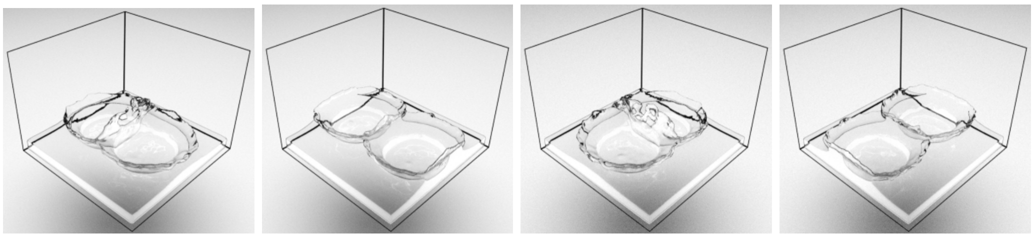


Figure 3.20: *The sampling density of the input simulations that are used for the training impacts the resulting reconstruction quality. Here, we increasingly vary the number of initial liquid drops on the x-axes (top) and show the interpolation result (inside the red box).*

3.4.4 3D Liquid Examples

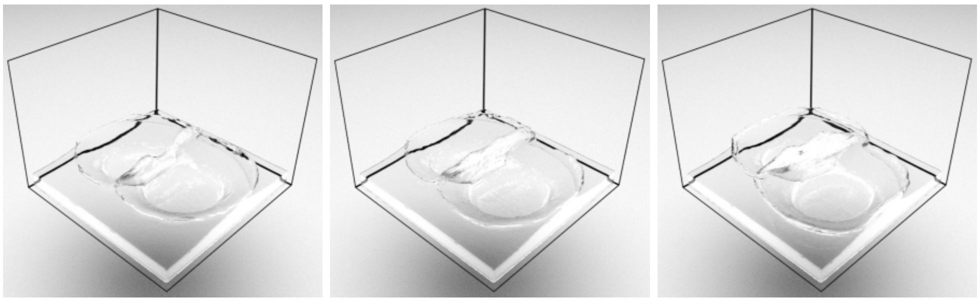
Spheres Dropping on a Basin We demonstrate that our approach can also handle splashing liquids. We use a setup for two spheres dropping on a basin, which is parameterized by the initial distance of the spheres, as well as by the initial drop angles along the $XZ-$plane relative to the basin. We sample velocity field sequences by combining 5 different distances and 10 different angles; Figure 3.21 shows 4 of the training samples. With 150 frames in time, the network is trained with $7,500$ unique velocity fields. We used a single-phase solver and extrapolated velocities from the liquid to the air phase before training (extrapolation size = 4 cells). Figure 3.22, middle, shows our result for an interpolated angle of $\hat{\theta} = 9°$ and a sphere distance of $\hat{d} = 0.1625$, given two CNN-reconstructed input samples on the left $(\theta = 0°, d = 0.15)$ and right $(\theta = 18°, d = 0.175)$. Our results demonstrate



$d = 0.15, \theta = 0°$ $d = 0.25, \theta = 0°$ $d = 0.15, \theta = 90°$ $d = 0.25, \theta = 90°$

Figure 3.21: *Training samples for the liquid spheres scene. In total we used* 50 *simulation examples with varying distances and angles.*

that the bulk of the liquid dynamics are preserved well. Small scale details such as high-frequency structures and splashes, however, are particularly challenging and deviate from the reference simulations.



$d = 0.15, \theta = 0°$ $\hat{d} = 0.1625, \hat{\theta} = 9°$ $d = 0.175, \theta = 18°$

Figure 3.22: *CNN-generated results with parameter interpolation for the liquid spheres example. While the far left and right simulations employ parameter settings that were part of the training data, the middle example represents a new in-between parameter point which is successfully reconstructed by our method.*

Viscous Dam Break In this example, a dam break with four different viscosity strengths ($\mu = 2 \times [10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$) was used to train the network. Our method can reconstruct simulations with different viscosities accurately, and also interpolate between different viscosities with high fidelity. In Figure 3.23, the CNN-reconstructed, green-colored liquids have direct correspondences in the original dataset; the pink-colored simulations are interpolation results between the two nearest green samples. Additionally, it is also possible to increase the viscosity over time as shown in Figure 3.24. The results show that this works reliably although the original parameterization does neither support time-varying viscosities nor do the training samples represent such behavior.

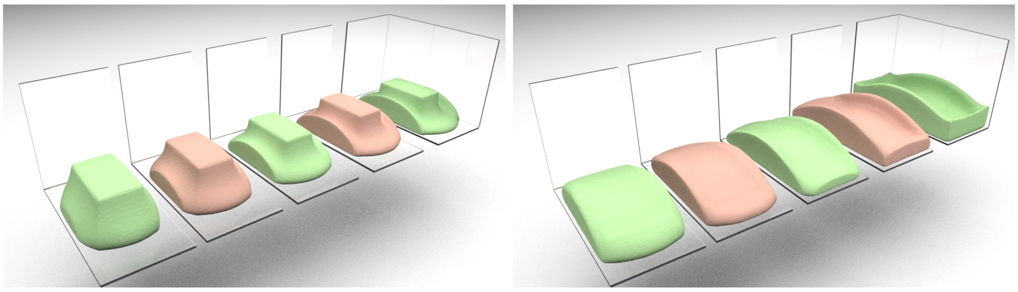


Figure 3.23: *Snapshots of a CNN reconstructed dam break with different viscosity strengths for two different frames. Green liquids denote correspondences with ground truth ($\mu = 2 \times [10^{-4}, 10^{-3}, 10^{-2}]$, back to front) while pink ones are interpolated ($\hat{\mu} = 2 \times [5^{-3}, 5^{-2}]$, back to front).*

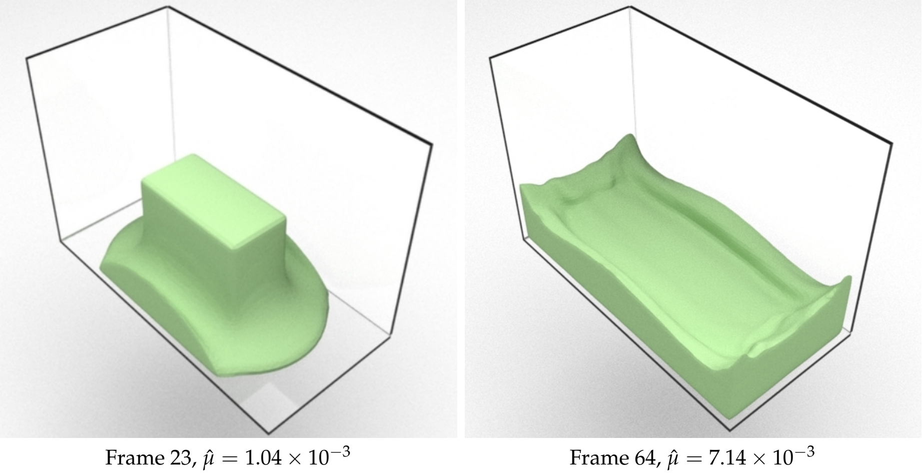

Frame 23, $\hat{\mu} = 1.04 \times 10^{-3}$
Frame 64, $\hat{\mu} = 7.14 \times 10^{-3}$

Figure 3.24: *Reconstruction result using a time varying viscosity strength. In the first few frames the liquid quickly breaks into the container. As the simulation advances, the viscosity increases and the liquid sticks to a deformed configuration.*

Slow Motion Fluids Our supplemental video additionally shows an interesting use case that is enabled by our CNN-based interpolation: the generation of temporally upsampled simulations. Based on a trained model we can create slow-motion effects, which we show for the liquid drop and dam break examples.

3.5 Evaluation and Discussion

3.5.1 Training

Our networks are trained on normalized data in the range $[-1, 1]$. In case of velocity fields, we normalize them by the maximum absolute value of the entire dataset. We found that batch or instance normalization techniques [Ulyanov et al., 2016] do not improve our velocity fields output, as the highest (lowest) pixel intensity and mean deviation might vary strongly within a single batch. Frames from image-based datasets have a uniform standard deviation, while velocity field snapshots can vary substantially. Other rescaling techniques, such as standardization or histogram equalization, could potentially further improve the training process.

Convergence of the Training The presented architecture is very stable and all our tests have converged reliably. Training time highly depends on the example and the targeted reconstruction quality. Generally, 3D liquid examples require more training iterations (up to 100 hours of training) in order to get high quality surfaces, while our smoke examples finished on average after 72 hours of training.

Figure 3.25 shows a convergence plot of the 2D smoke example, with training iterations on the x-axis and error on the y-axis. The superimposed images show clearly how quality increases along with training iterations. After about $180,000$ iterations, the smoke plume is already reconstructed with good accuracy. This corresponds to roughly 3 hours of training on our hardware. Scaling tests with various 2D grid resolutions (64×48, 128×96, 256×192, 512×384) have shown that the training speed scales proportionally with the resolution, while keeping the mean absolute error at a constant level.

3.5.2 Performance Analysis

Table (3.1) summarizes the statistics of all presented examples. In terms of wall-clock time, the proposed CNN approach generates velocity fields up to 700× faster than re-simulating the data with the underlying CPU solver. Some care must be taken when interpreting this number because our *tensorflow* [Abadi et al., 2016] network runs on the GPU, while the original *mantaflow* code runs on the CPU. Fluid simulations are known to be memory bandwidth-limited [Kim, 2008], and the bandwidth discrepancy between a GTX 1080 (320 GB/s) and our Intel desktop (25.6 GB/s) is a factor of 12.5.

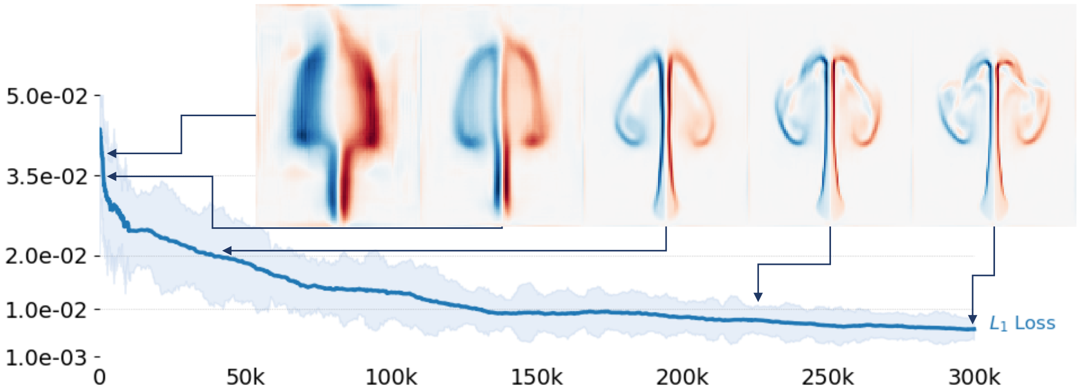


Figure 3.25: *Convergence plot of the L_1 loss for the 2D smoke sequence from Figure 3.8.*

However, even if we conservatively normalize by this factor, our method achieves a speed-up of up to 58×. Thus, the core algorithm is still at least an order of magnitude faster. To facilitate comparisons with existing subspace methods, we do not include the training time of our CNN when computing the maximum speedup, as pre-computation times are customarily reported separately. Instead, we include them in the discussion of training times below.

Contrary to traditional solvers, our approach is able to generate multiple frames in time independently. Thus, we can efficiently concatenate CNN queries into a GPU batch, which then outputs multiple velocity fields at once. Adding more queries increases the batch size (Table (3.1), 5^{th} column, number in brackets), and the maximum batch size depends on the network size and the hardware's memory capacity. Since we are using the maximum batch size possible for each scene, the network evaluation time scales inversely with the maximum batch size supported by the GPU. Due to the inherent capability of GPUs to efficiently schedule floating point operations, the time for evaluating a batch is independent of its size or the size of the network architecture. Additionally, our method is completely oblivious to the complexity of the solvers used to generate the data. Thus, more expensive stream function [Ando et al., 2015a] or energy-preserving [Mullen et al., 2009] solvers could potentially be used with our approach, yielding even larger speed-ups.

In contrast, computing the linear basis using traditional SVD-based subspace approaches can take between 20 [Kim and Delaney, 2013] and 33 [Wicke et al., 2009] hours. The process is non-iterative, so interrupting the computation can yield a drastically inferior result, i.e. the most important singular vector may not have been discovered yet. [Stanton et al., 2013] reduced the pre-computation time to 12 hours, but only by using a 110-node

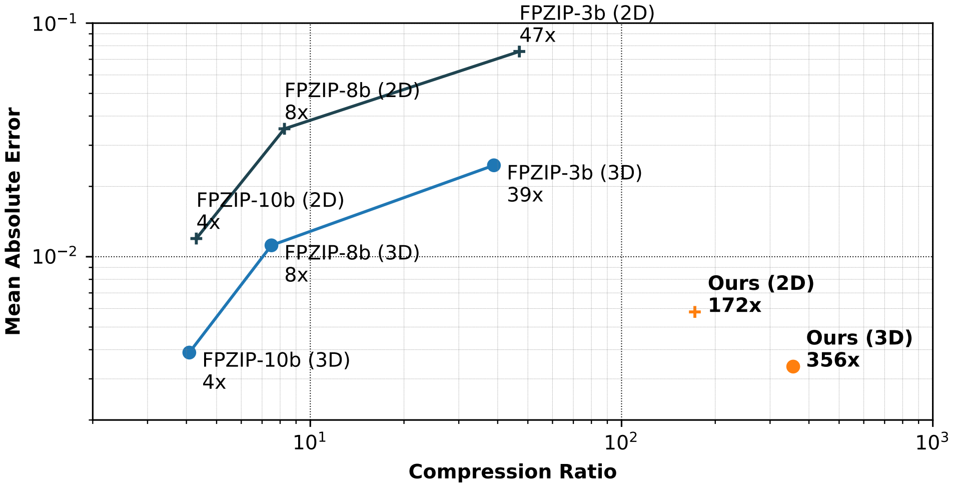
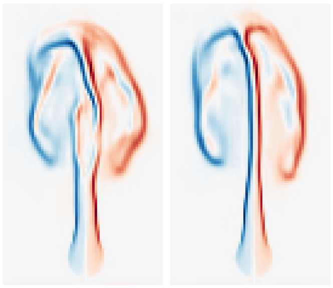


Figure 3.26: *Compression ratio and mean absolute error plot of FPZIP and ours. The postfix of FPZIP label represents the number of bits of precision.*

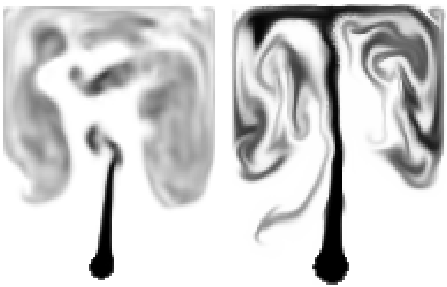
cluster. In contrast, our iterative training approach is fully interruptible and runs on a single machine.

Compression The memory consumption of our method is at most 30 MB, which effectively compresses the input data by up to 1300×. Previous subspace methods [Demby Jones et al., 2016] only achieved ratios of 14×, hence our results improve on the state-of-the-art by two orders of magnitude. We have compared the compression ability of our network to FPZIP [Lindstrom and Isenburg, 2006], a data reduction technique often used in scientific visualization [Li et al., 2018b] as shown in Figure 3.26. FPZIP is a prediction-based compressor using Lorenzo predictor, which is designed for large floating-point datasets in arbitrary dimension, and the number of precision bits is the only parameter of it. In Figure 3.26, each label shows the number of bits and the dimension of a dataset with its achieved compression rate. For the two datasets of Section 3.4.1 and Section 3.4.2, we reached a compression of 172× and 356×, respectively. In comparison, FPZIP achieves a compression of 4× for both scenes with 16 bits of precision (with a mean absolute error comparable to our method). When allowing for a 6× larger mean absolute error, FPZIP achieves a 47× and 39× compression. I.e., the data is more than 4× larger and has a reduced quality compared to our encoding. Thus, our method outperforms commonly used techniques for compressing scientific data.

3.5.3 Quality of Reconstruction and Interpolation



Training Data Several factors affect the reconstruction and interpolation quality. An inherent problem of machine learning approaches is that quality strongly depends on the data used for training. In our case, the performance of our generative model for interpolated positions is sensitive to the input sampling density and parameters. If the sampling density is too coarse, or if the output abruptly changes with respect to the variation of parameters, errors may appear on reconstructed velocity fields. These errors include the inability to accurately reconstruct detailed flow structures, artifacts near obstacles, and especially ghosting effects in the interpolated results. An example of ghosting is shown in the inset image where only 11 training samples are used (left), instead of the 21 (right) from Section 3.4.1.



Target Quantities We have also experimented with training directly with density values (inset image, left) instead of the velocity fields (inset image, right). In case of density-trained networks, the dynamics fail to recover the non-linear nature of momentum conservation and artifacts appear. Advecting density with the reconstructed velocity field yields significantly better results. A more detailed discussion about the quality of the interpolation regarding the number of input samples and discrepancies between velocity and density training is presented in the Section 3.4.1.

Latent Space Size We performed additional tests regarding the reconstruction quality of our autoencoder network in the Section 3.3 relative to the number of parameters used for the latent space encoding. As shown in Figure 3.27, the quality of the reconstruction increases with the number of dimensions used for the latent space reconstruction. In Figure 3.28, we plot training losses for the different dimensions of Figure 3.27. Notice that relatively small differences of the L1 loss impact the visual results significantly for smoke profiles.

Conversely, the latent space integration network performed better with a smaller latent space size, as shown in Figure 3.28. Increasing the window size w reduces long-term errors of the time advancement. We show the error plots for three different window sizes (1, 5, 30) in Figure 3.29 (left). In Figure 3.29 (right) we show advected densities by the latent space integrator network trained with $w = 1$ and $w = 30$. We notice that the errors of the advection are larger for the window size equal to 1, and some smoke samples hang mid-air (highlighted by the red square).

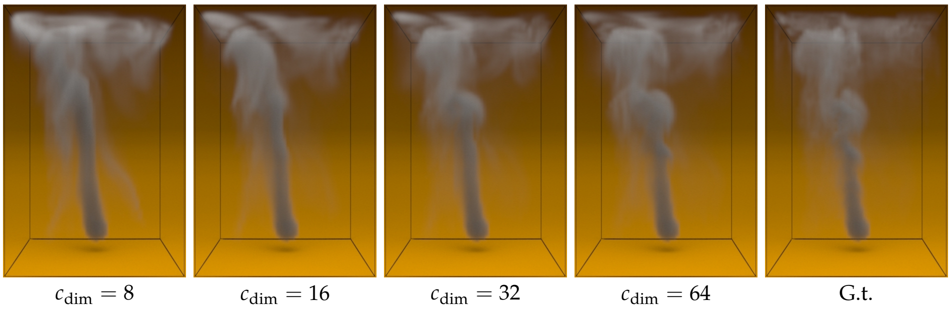


Figure 3.27: *Comparisons of the quality of autoencoder reconstruction with varying sizes for the latent space c_{dim}. As the latent space dimensionality increases, the quality of the reconstructed smoke plume also increases.*

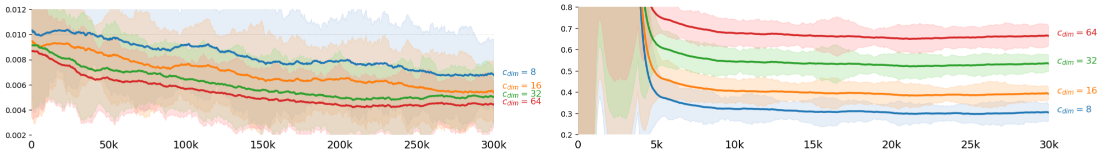


Figure 3.28: *L_1 loss plot of the autoencoder training (left) and L_2 loss plot of the latent space integration network training (right). Increasing the dimension of the latent space c_{dim} reduces reconstruction errors, while it degrades the accuracy of the integration.*

Velocity Loss A comparison between our compressible loss, incompressible functions and ground truth is shown in Figure 3.30. The smoke plume trained with the incompressible loss from Equation (3.1) shows a richer density profile closer to the ground truth, compared to results obtained using the compressible loss.

Figure 3.31 shows the velocity divergence evolution over the network iterations for the 2D smoke example when the incompressible loss is *not* applied. The orange and red lines show errors at ground truth positions without and with the gradient loss (i.e., $\lambda_{\nabla \mathbf{u}} = 0/1$ with label *Disc.*), respectively. The blue and green line labeled *Cont.* correspond to the measured divergence at interpolated positions. The velocity gradient clearly helps the network to reduce divergence in the generated flow fields.

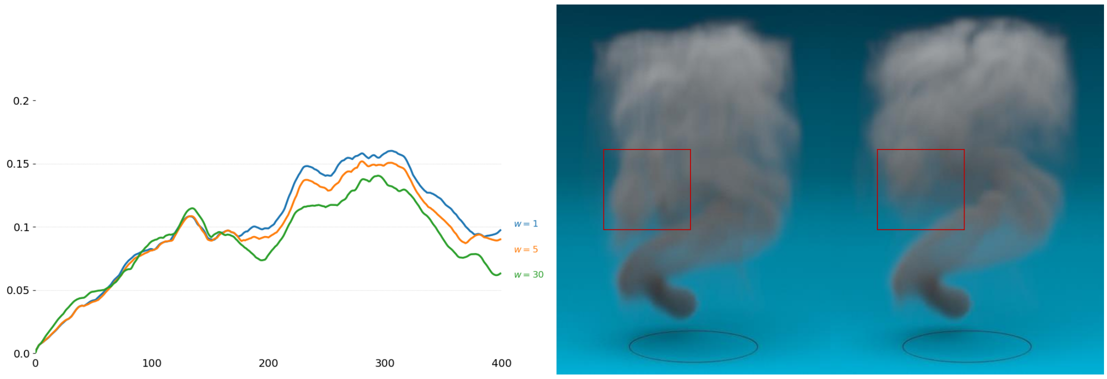


Figure 3.29: *We show the effect of the training with varying sizes of windows (1, 5 and 30) for the cyclic smoke example. On the left, the L_2 loss plot of the latent space integration network is shown. On the right, we show the visual impact for window sizes of 1 and 30, respectively. Note that there are densities left hanging in mid air in the red box, which are not advected properly due to wrong velocity states recovered by the latent space integration network.*

Additionally, we compared the performance of the gradient loss on smooth stream function and pressure fields ((a) and (b) of Figure 3.32, respectively). Since both are considerably smoother than velocity fields, more high-frequency network features are in the null-space of the L_1 loss minimization. Although not visually obvious when directly comparing stream function and pressures field values (Figure 3.32, top), artifacts clearly appear for their gradients (Figure 3.32, bottom). These gradients significantly influence simulation results, e.g., visible in the advected densities shown in (Figure 3.33).

Boundary Conditions The proposed CNN is able to handle immersed obstacles and boundary conditions without additional modifications. Figure 3.34 shows sliced outputs for the scene from Figure 3.12 which contains a sphere obstacle. We compare velocity (top) and vorticity magnitudes (bottom). The first and last images show the reconstruction of the CNN for p_x positions that have correspondences in the training dataset. The three images in the middle show results from linearly blending the closest velocity fields, our CNN reconstruction and the ground truth simulation, from left to right respectively. In the case of linearly blended velocity fields, ghosting arises as features from the closest velocity fields are super-imposed [Thuerey, 2016], and the non-penetration constraints for the obstacle are not respected, as velocities are presented inside the intended obstacle positions. In Figure 3.35, we plot the resulting velocity penetration errors. Here we compute the mean absolute values of the velocities inside the voxelized sphere, nor-

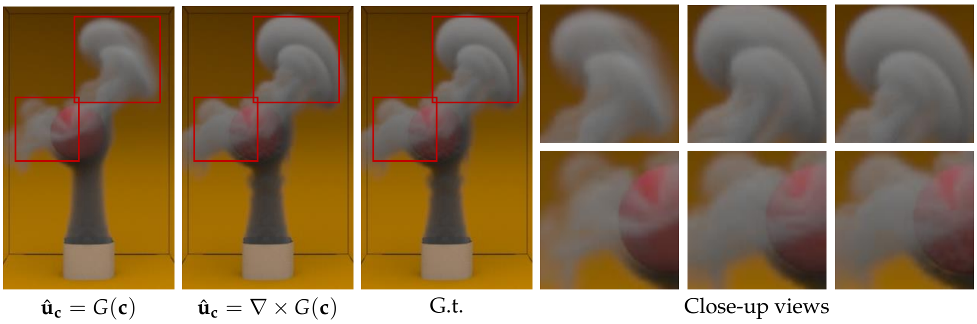

$\hat{\mathbf{u}}_\mathbf{c} = G(\mathbf{c})$
$\hat{\mathbf{u}}_\mathbf{c} = \nabla \times G(\mathbf{c})$
G.t.
Close-up views

Figure 3.30: *Comparisons of the results from networks trained on our compressible loss, incompressible loss and the ground truth, respectively. On the right sequence we show the highlighted images from the simulations on the left. We notice that the smoke patterns from the incompressible loss are closer to ground truth simulations.*

malized by the mean sum of the velocity magnitudes for all cells around a narrow band of the sphere. Boundary errors are slightly higher for interpolated parameter regions (orange line in Figure 3.35), since no explicit constraint for the object's shape is enforced. However, the regularized mean error still accounts for less than 1% of the maximum absolute value of the velocity field. Thus, our method successfully preserves the non-penetration boundary conditions.

Liquid-air Interface Due to the separate advection calculation for particles in our FLIP simulations, smaller splashes can leave the velocity regions generated by our CNNs, causing surfaces advected by reconstructed velocities to hang in mid-air. Even though the reconstructed velocity fields closely match the ground truth samples, liquid scenes are highly sensitive to such variations. We removed FLIP particles that have smaller velocities than a threshold in such regions, which was sufficient to avoid hanging particles artifacts.

3.5.4 Extrapolation and Limitations

Extrapolation with Generative Model We evaluated the extrapolation capabilities for the case where only the generative part of our Deep Fluids CNN (Section 3.2) is used. Generally, extrapolation works for sufficiently small increments beyond the original parameter space. Figure 3.36 shows an experiment in which we used weights that were up to 30% of the original parameter range ($[-1, 1]$). The leftmost images show the vorticity plot

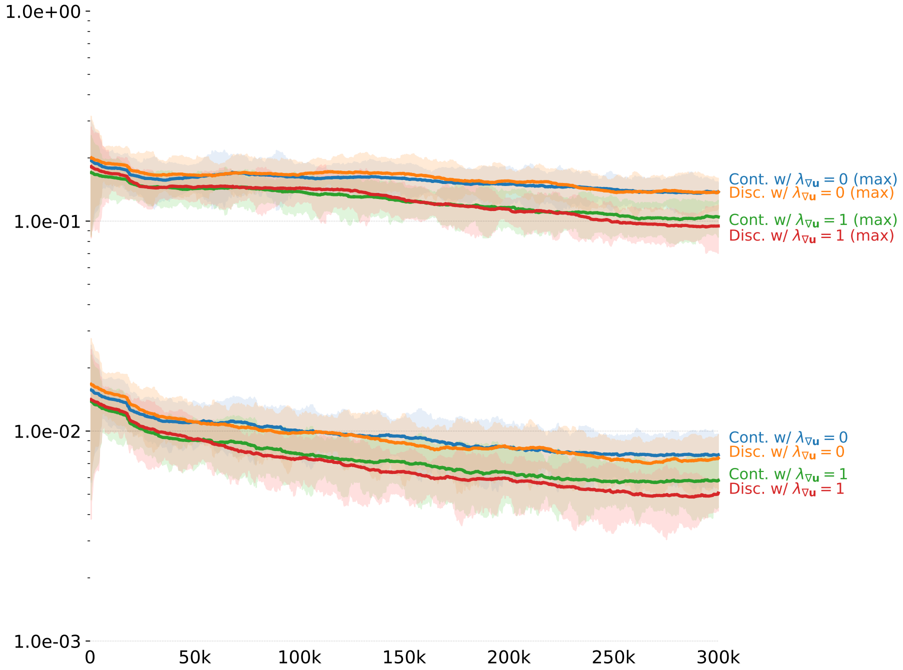


Figure 3.31: *Divergence max (top) and average (bottom) over number of iterations. Using the gradient velocity in the loss function ($\lambda_{\nabla\mathbf{u}} = 1$) reduces the divergence error, increasing reconstruction quality.*

for the maximum value of the range for the position (top), inflow size (middle), and time (bottom) parameters of the 2D smoke plume example. The rightmost images show the maximum variation of parameters, in which the simulations deteriorate in quality. In practice, we found that up to 10% of extrapolation still yielded plausible results.

Limitations Our Deep Fluids CNN is designed to generate velocity fields for parameterizable scenes. As such, our method is not suitable for reconstructing arbitrary velocity fields of vastly different profiles by reduction to a shared latent representation. As discussed in Section 3.5.3, there is also no enforcement of physical constraints such as boundary conditions for intermediate interpolated parameters. Thus, the capability of the network to reconstruct physically accurate samples on interpolated locations depends on the proximity of the data samples in the parameter space. Additionally, the reconstruction quality of the autoencoder and latent space integration networks are affected by the size of the latent space $\mathbf{c}$, and there is a possible issue of temporal jittering because of lack of the gradient loss on Equation (3.5).

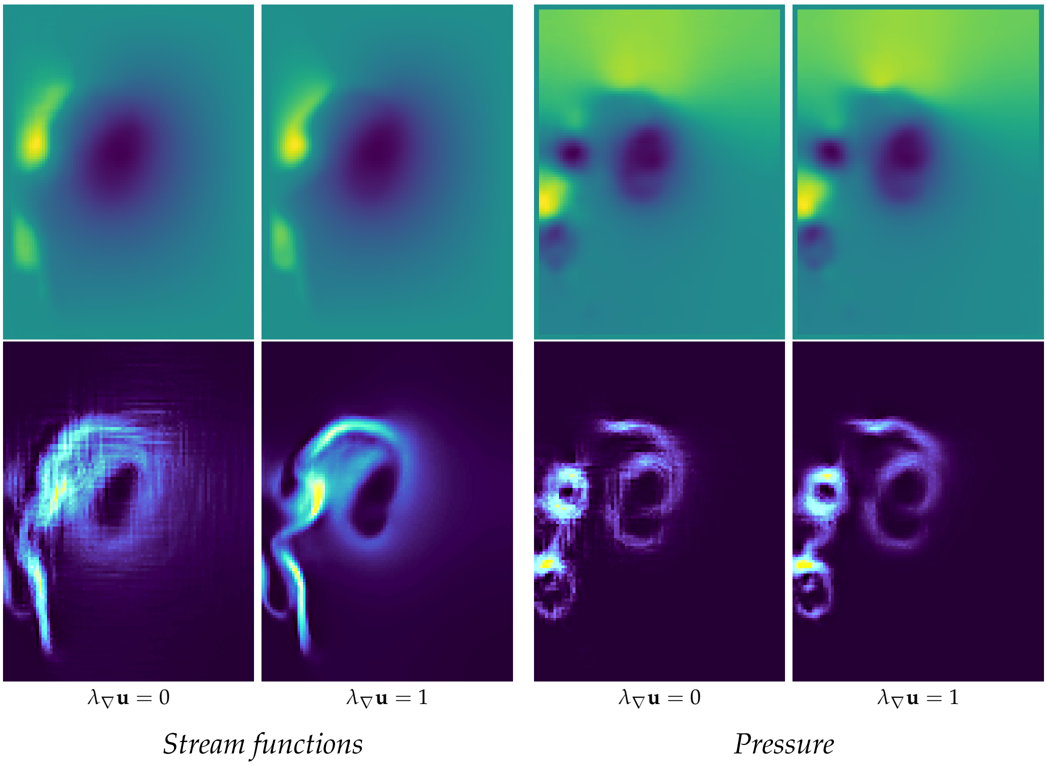


Figure 3.32: *Stream functions (a) and pressure (b) fields plots. Top images show original fields, while bottom ones show their gradients. We highlight that artifacts appear on those datasets when not using the gradient loss function ($\lambda_{\nabla\mathbf{u}} =$ 0).*

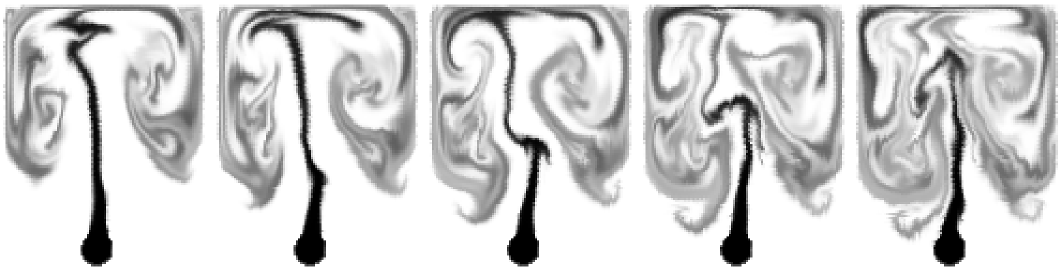


Figure 3.33: *Density plots advected by velocity fields from reconstructed streamfunctions of the network trained without gradient loss. Jagged artifacts are visible, e.g., near the central stem of the smoke plume.*

Scene	Grid Resolution	# Frames	Simulation Time (s)	Eval. Time (ms) [Batch]	**Speed Up** (×)	Dataset Size (MB)	Network Size (MB)	**Compression Ratio**	Training Time (h)
Smoke Plume	96 × 128	21,000	0.033	0.052 [100]	635	2064	12	172	5
Smoke Obstacle	64 × 96 × 64	6,600	0.491	0.999 [5]	513	31143	30	1038	74
Smoke Inflow	112 × 64 × 32	3,750	0.128	0.958 [5]	128	10322	29	356	40
Liquid Drops	96 × 48 × 96	7,500	0.172	1.372 [3]	125	39813	30	**1327**	134
Viscous Dam	96 × 72 × 48	600	0.984	1.374 [3]	**716**	2389	29	82	100
Rotating Smoke	48 × 72 × 48	500	0.08	0.52 [10]	308	995	38	26	49
Moving Smoke	48 × 72 × 48	80,000	0.08	0.52 [10]	308	159252	38	4191*	49

Table 3.1: *Statistics for training datasets and our CNN. Note that simulation excludes advection and is done on the CPU, while network evaluation is executed on the GPU with batch sizes noted in brackets. In case of liquids, the conjugate gradient residual threshold is set to* $1e^{-3}$*, while for smoke it is* $1e^{-4}$*. For the Rotating and Moving Smoke scenes, the numbers for training time and network size include both the autoencoder and latent space integration networks.*

** We optimize the network for subspace simulations rather than the quality of reconstruction, so we do not take this number into account when evaluating the maximal compression ratio.*

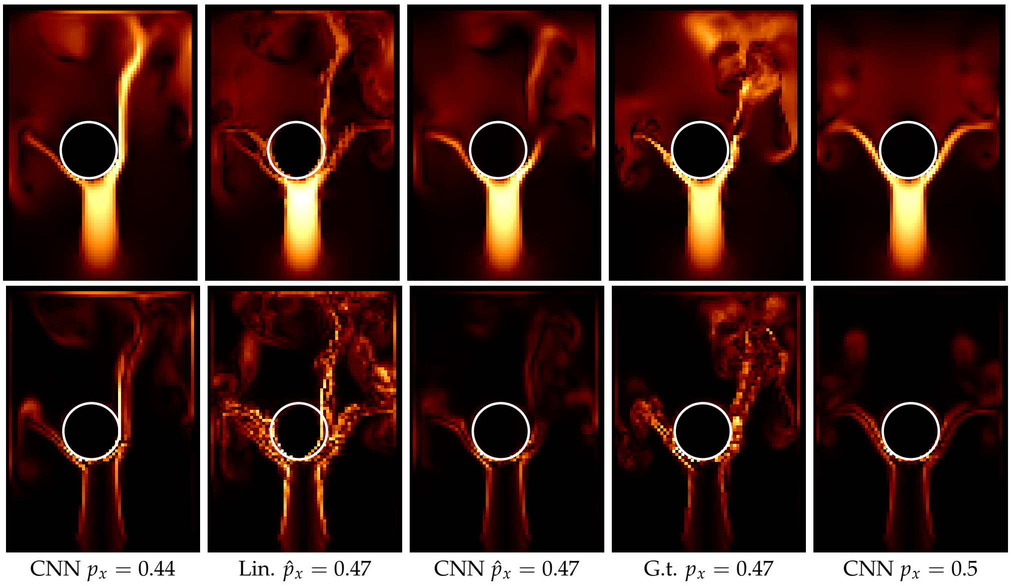


Figure 3.34: *Slice views of the last row of Figure 3.12. The color code represents the velocity (top) and vorticity (bottom) magnitudes. The second column shows a linear interpolation of the input. Despite the absence of any constraints on boundary conditions, our method (third column) preserves the shape of the original sphere obstacle, and yields significantly better results than the linear interpolation.*

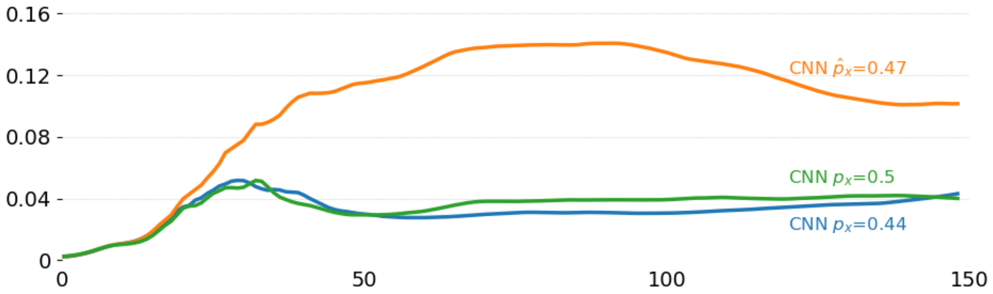


Figure 3.35: *Mean absolute error plot of velocity penetration for the smoke obstacle example. Though errors in interpolated samples are a bit higher than those of reconstructed samples, they do not exceed 1% of the maximum absolute value of the dataset.*

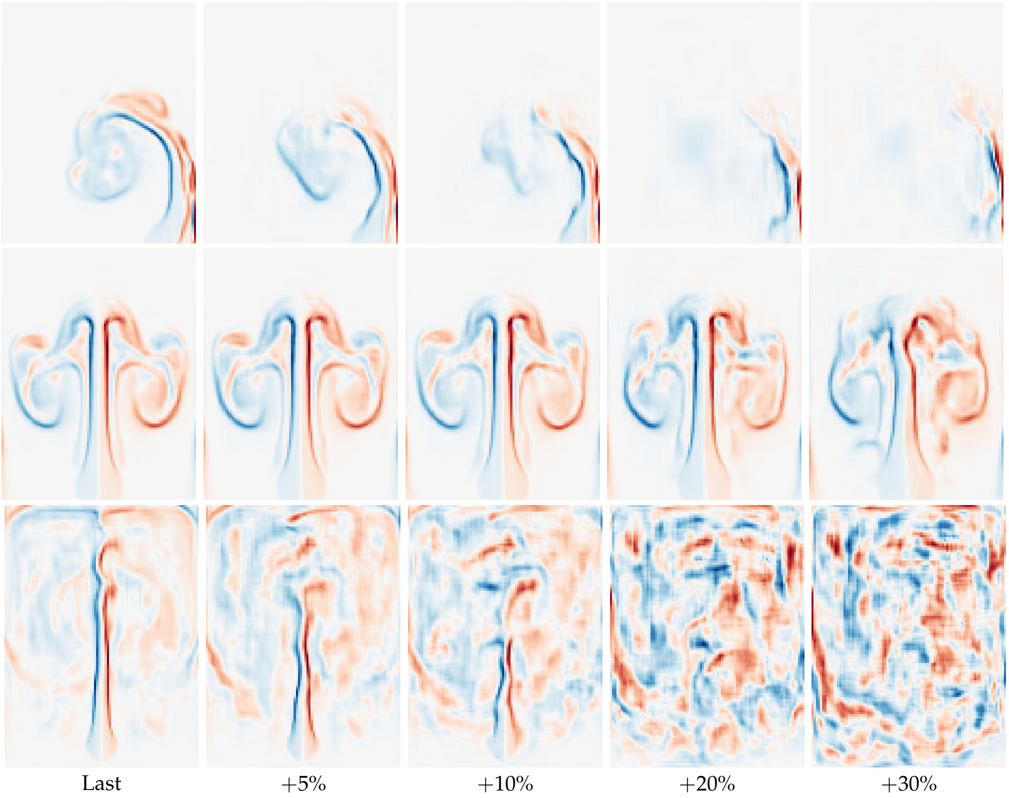


Figure 3.36: *2D smoke plume extrapolation results (from t. to b.: position, inflow width, time) where only the generative network is used. Plausible results can be observed for up to 10% extrapolation.*

CHAPTER 4

Neural Artistic Control of Smoke Simulations

This chapter is based on the following publication by [Kim et al., 2019a]:

B. Kim, V. C. Azevedo, M. Gross, B. Solenthaler, **Transport-Based Neural Style Transfer for Smoke Simulations**, *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2019)*, 38(6), Nov. 2019. (selected for the video trailer)

The supplemental video can be found here: `https://www.youtube.com/watch?v=67qVRhoOQPE`.

4.1 Overview

In the previous Chapter 3, we presented the first generative deep learning model that profitably constructs a wide variety of fluid behaviors, from turbulent smoke to viscous liquids. Although our method allows many applications such as fluid motion reconstruction, interpolation, latent space simulation and provides profound insights into flow data, it does not focus on the artistic control capabilities on fluids.

In this Chapter 4, we propose a novel method to synthesize semantic structures onto volumetric flow data by taking advantage of the simple yet powerful machinery developed for image editing, inspired by Neural Style Transfer (NST) methods for images [Gatys et al., 2016a] and meshes [Kato

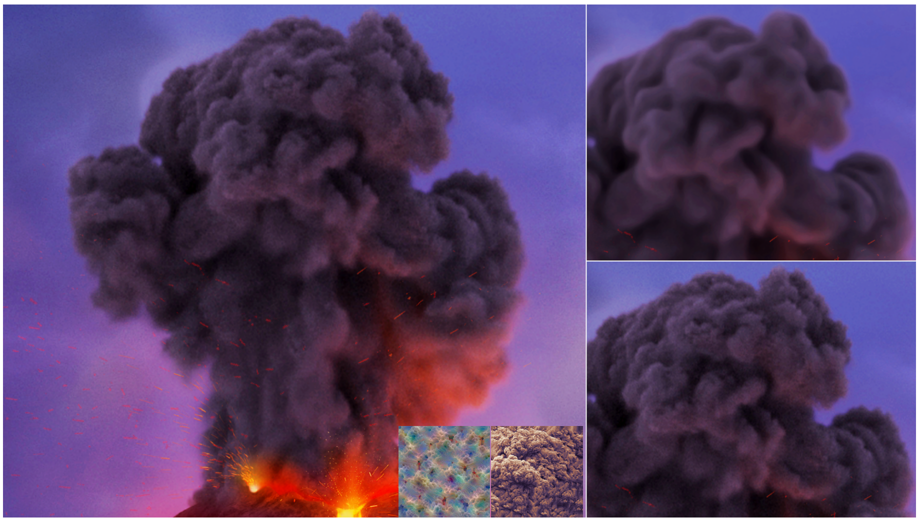


Figure 4.1: *Volcanic smoke simulation. Left: stylized output by our transport-based neural style transfer; right: close-up views of low-resolution base input and ours with a cloud motif [gfv, 2015] and a smoke exemplar ©Richard Roscoe via [Jamriška et al., 2015].*

et al., 2018; Liu et al., 2018b]. We modify 3D density fields by combining individual 2D stylizations from multiple views, which are synthesized by matching features of a pre-trained Convolutional Neural Network (CNN). Since the CNN is trained for image classification tasks, a vast library of patterns and class semantics is available, enabling novel content-aware flow manipulations that range from transferring low (edges and patterns) to high (complex structures and shapes) level features from images to smoke simulations (Figures 4.1, 4.12 and 4.13). In this way, our method allows for automatic instantiation of structures in flow regions that naturally share features with a given target pattern or semantic class.

Crucially, and in contrast to existing NST methods, our style transfer algorithm is physically inspired. It computes a velocity field that stylizes a smoke density with an input target style, yielding results that naturally model the underlying transport phenomena. To improve temporal consistency, we propose a method which aligns stylization velocities from adjacent frames, enabling the control of how smoothly stylized structures change in time. To handle volumetric smoke stylization, multiple stylized 2D views are seamlessly combined into a 3D representation, resulting in coherent stylized smoke structures from arbitrary camera viewpoints. The proposed method is end-to-end differentiable and it can be readily optimized by gradient descent approaches. This is enabled by a novel volumetric differentiable smoke renderer, which is tailored for stylization purposes. Our results

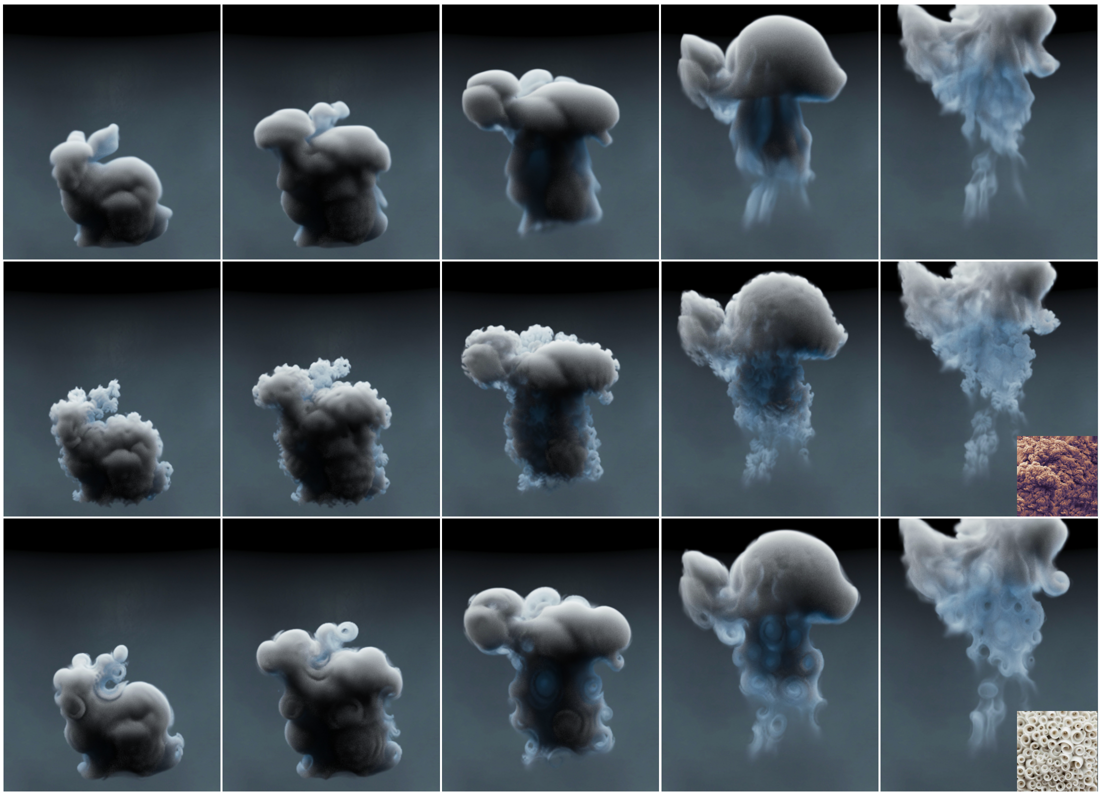


Figure 4.2: *Frames of a style transfer smoke example: base simulation (top), stylized output with volcano (middle, ©Richard Roscoe) and spiral [spi, 2015](bottom) images.*

demonstrate that our method captures a wide spectrum of different styles and high-level semantics, and hence can be used to transfer patterns and regular structures, turbulence effects, shapes and artistic styles onto existing simulations.

4.2 Transport-Based Neural Style Transfer

Our method employs pre-trained CNNs for natural image classification as both feature extractor and synthesizer. As an alternative, we considered CNNs trained on synthetic 3D representations such as voxels [Wu et al., 2015], meshes [Masci et al., 2015] and point clouds [Qi et al., 2017]. However, CNNs trained on 2D natural images have seen richer and denser information as there is a more expressive incidence of high-frequency features [Qi et al., 2016], and datasets have been thoroughly analyzed in terms of interpretability. Thus, as a classification CNN gets deeper, it shows its hier-

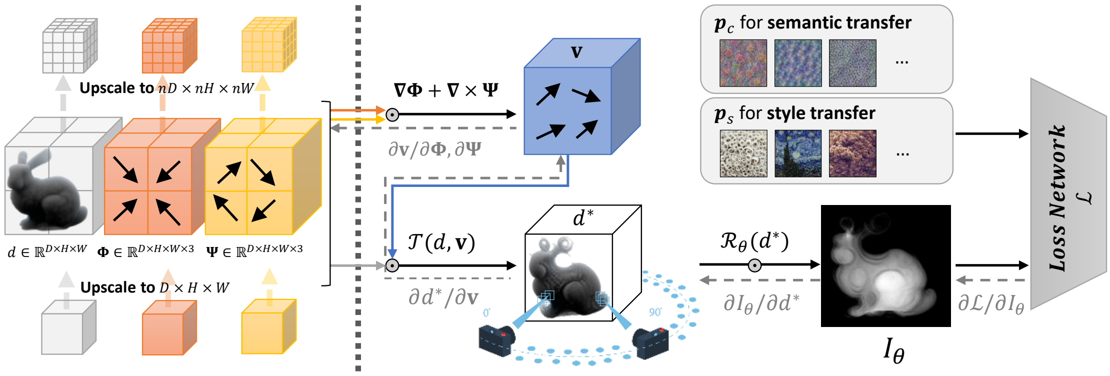

Upscale to $nD \times nH \times nW$
$d \in \mathbb{R}^{D \times H \times W}$
$\Phi \in \mathbb{R}^{D \times H \times W \times 3}$
$\Psi \in \mathbb{R}^{D \times H \times W \times 3}$
Upscale to $D \times H \times W$
$\nabla\Phi + \nabla \times \Psi$
$\mathbf{v}$
$\partial\mathbf{v}/\partial\Phi, \partial\Psi$
$\mathcal{T}(d, \mathbf{v})$
$\partial d^*/\partial\mathbf{v}$
d^*
0°
90°
p_c for semantic transfer
...
p_s for style transfer
...
$\mathcal{R}_\theta(d^*)$
$\partial I_\theta/\partial d^*$
I_θ
$\partial\mathcal{L}/\partial I_\theta$
Loss Network
$\mathcal{L}$

Figure 4.3: *Pipeline of our TNST method. On the left, three input fields d, Φ, Ψ for the stylization algorithm are shown. Φ, Ψ are iteratively updated during the optimization, while the stylized density output is represented by d^*. All fields are firstly downsampled and stylized on their coarser representations, so features can be enhanced through larger regions of the smoke. Cubic upsampling is performed on d, Φ, Ψ and the stylization runs again on a finer resolution; this process repeats until the specified resolution is matched. The right side of the diagram illustrates how our optimization works. The black arrows show the direction of a feed-forward pass from potentials Φ, Ψ to the loss network, and the gray arrows represent the backpropagation path for computing gradients.*

archical interpretation of natural images organized from low-level patterns to high-level semantics [Olah et al., 2017].

The original neural style transfer (NST) [Gatys et al., 2016a] transforms an initial noise image I to match the content (I_c) and style (I_s) of input target images. The content loss $\mathcal{L}_c$ measures selected filter responses from a pre-trained classification CNN, while the style loss $\mathcal{L}_s$ measures the difference between specific filter's statistical distributions. The neural style transfer solves the optimization of

$$\hat{I} = \arg\min_{I} \alpha\mathcal{L}_c(I, I_c) + \beta\mathcal{L}_s(I, I_s), \tag{4.1}$$

where the weights α and β control how the content and style modify the initial image I along the optimization process.

Applying existing NST methods to stylize smoke data will lead to arbitrary creation of sources, since the volumetric density field is evaluated as an intensity image. Thus, we propose a transport-based neural style transfer (TNST), in which the stylization is driven by velocity fields instead of direct pixel / voxel corrections as introduced in Figure 4.3. The transport-based

Table 4.1: *Symbols, operators and configurable parameters.*

$\mathcal{L}_c, \mathcal{L}_s$	Content and style losses
α, β	Weights controlling content and style losses
$d, \mathbf{u}$	Input density and simulation velocity field
$d^*, \mathbf{v}$	Stylized density and stylization velocity field
σ	Density integrated spatially for a single frame
$\Phi, \mathbf{\Psi}$	Irrotational and incompressible potentials
λ	Weight between irrotational and incompressible vector fields
$\mathbf{p}_c, \mathbf{p}_s$	Content and style input parameters
$\mathcal{R}, \mathcal{T}$	Rendering and advection operators
$\mathcal{F}^l, \hat{\mathcal{F}}^l$	CNN's spatial and flattened feature maps for layer l
$\mathcal{M}^l$	User-defined feature map at layer l for semantic transfer
H, W, C	Height, width and channels of feature map or image
ω, w	Temporal coherency weight and window size
γ	Transmittance absorption factor
θ, Θ	Individual viewpoint and set of viewpoints
η	Learning rate size

approach yields more degrees of freedom than directly changing the densities; specifically, it will yield a vector field while a standard value-based approach will output a scalar field correction. This is particularly useful in 3D, since the directional information encoded by the vector field will be used to merge stylizations from distinct camera viewpoints. Additionally, this approach enables the control over smoke density sources and sinks during stylization: we implement a divergence control through the decomposition of the stylization vector field into its incompressible and irrotational parts. A comparison between value- and velocity-based stylizations is shown in Figure 4.4.

4.2.1 Single-Frame Multi-View Stylization

We define a single-frame loss for a given input image $I \in \mathbb{R}^{H \times W}$

$$\mathcal{L}(I, \mathbf{p}_c, \mathbf{p}_s) = \sigma \left[\alpha \mathcal{L}_c(I, \mathbf{p}_c) + \beta \mathcal{L}_s(I, \mathbf{p}_s) \right], \tag{4.2}$$

where $\sigma = \sum_i^H \sum_j^W I_{ij}$ is the per-pixel intensity ($I_{ij} \in [0, 1]$) integrated over the image, and $\mathbf{p}_c$ and $\mathbf{p}_s$ are user-specified parameters (Sections 4.2.2 and 4.2.3) that control content and style transfers. We normalize the loss function by the integrated pixel intensities, since, contrary to natural images, pixels

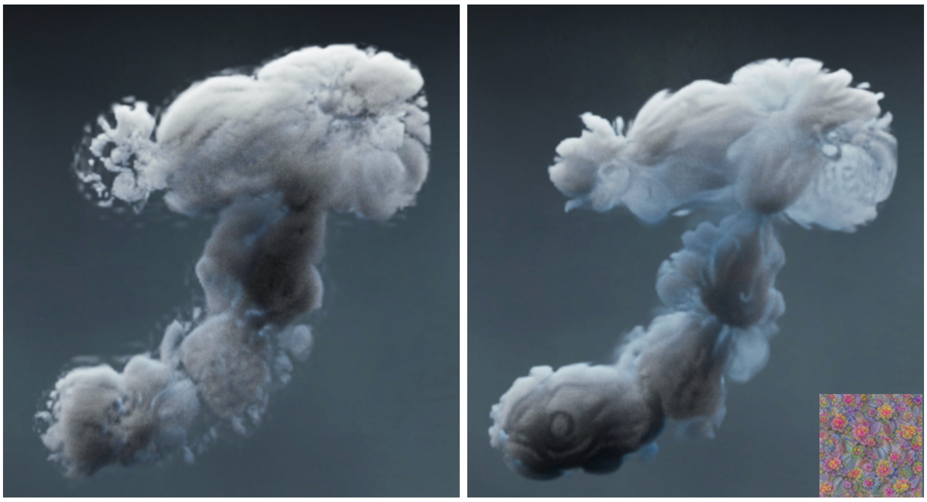


Figure 4.4: *Value-based (left) against transport-based density optimization (right) with a flower motif [gfv, 2015]. The value-based approach used in traditional image stylization approaches produces ghosting artifacts and thinner smoke structures, since density sources can be created and removed to match targeted features.*

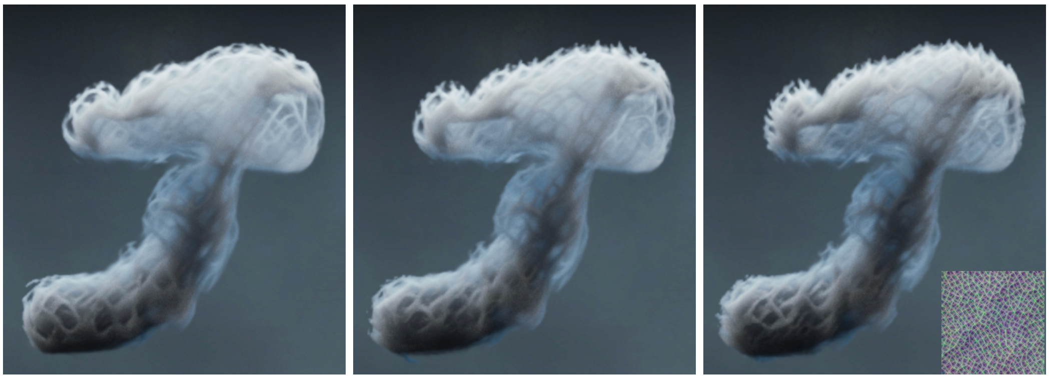


Figure 4.5: *Results from semantic transfer of a net structure [gfv, 2015]. Irrotational (left), mixed (middle) and incompressible (right) velocity fields.*

from our rendered depiction represent smoke intensities that will be used for stylization.

Given the input density field $d : \mathbb{R}^3 \rightarrow \mathbb{R}$ and a velocity field $\mathbf{v} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, the transport function $\mathcal{T}(d, \mathbf{v})$ advects d by $\mathbf{v}$. Unlike image-based stylization, where pixels already contain color information of the represented image, our approach has to compute a valid rendering of the flow data. The renderer $\mathcal{R}_\theta(d)$ outputs a grayscale image (Section 4.3) representing the density field for a specific viewpoint angle θ from a discrete set of viewpoints Θ. Our method optimizes a velocity field decomposed by a linear combination of its irrotational and incompressible parts by

$$\mathbf{v} = \lambda \nabla \Phi + (1 - \lambda) \nabla \times \mathbf{\Psi} \tag{4.3}$$

to achieve a desired stylized density field by minimizing

$$\hat{\Phi}, \hat{\mathbf{\Psi}} = \operatorname*{arg\,min}_{\Phi, \mathbf{\Psi}} \sum_{\theta \in \Theta} \mathcal{L}(\mathcal{R}_\theta(d^*), \mathbf{p}_c, \mathbf{p}_s), \tag{4.4}$$

where $d^* = \mathcal{T}(d, \mathbf{v})$ is the density field evolving towards stylization. Since our formulation optimizes for the scalar and vector potentials that transport the smoke, we allow the user to have direct control over the divergence of the stylization velocity field. Incompressible and irrotational velocity fields generate artistically different results and Figure 4.5 shows a comparison between both approaches. In order to create 3D structures, contributions from individual viewpoints $\mathcal{R}_\theta$ are summed, similarly to [Liu et al., 2018b]. We will discuss camera view sampling and renderer specifications in Section 4.3. The next sections describe the loss functions that we use for semantic ($\mathcal{L}_c$) and style transfers ($\mathcal{L}s$).

4.2.2 Semantic Transfer

Inspired by DeepDream [Mordvintsev, 2016], our method allows novel semantic transfer for stylizing smoke simulations by manipulating the content represented by the smoke. For example, smoke densities can be modified to portray patterns and shapes, such as squares or flowers, as depicted in Figure 4.12. Let $\mathcal{F}^l(I) \in \mathbb{R}^{H_l \times W_l \times C_l}$ denote a feature map of $[H_l, W_l]$ dimensions with C channels at the layer l of the network with respect to an input image I. The user-specified parameter $\mathbf{p_c}$ consists of an array of feature maps $\mathcal{M} \in \mathbb{R}^{H_l \times W_l \times C_l}$ for all layers $l \in L$ specified by the array. We then define the content loss as to match features of a density field rendered image to a user-defined feature map by

$$\mathcal{L}_c(I, \mathbf{p}_c) = \sum_l^L \left[\frac{1}{H_l W_l C_l} \sum_i^{H_l} \sum_j^{W_l} \sum_k^{C_l} \left(\mathcal{F}^l_{ijk}(I) - \mathcal{M}^l_{ijk} \right)^2 \right], \tag{4.5}$$

where $\mathcal{F}^l_{ijk}(I)$ denotes an activated neuron of the CNN respective to the input image I at position (i, j) of the feature map's k^{th} channel. The feature map $\mathcal{M}$ represents semantic features that will be transferred to the smoke (e.g., flowers), and it controls the abstraction level of structures created in the stylization process. Choosing a feature map that lies on deeper levels of the network will create more intricate motifs, as shown in Figure 4.12. The

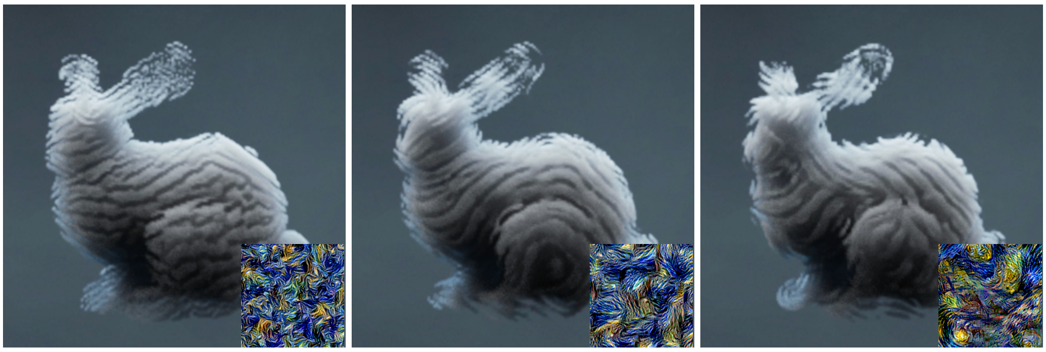


Figure 4.6: *Abstraction levels of style features and their impact on the smoke stylization result. We can control low (left), medium (center) and high (right) levels of features. The corner images show style representations corresponding to different feature levels.*

user can choose the abstraction level for the semantic transfer to match the specific content of an input image by selecting shallow levels of the network layers; or, conversely, match classification textual tags, enabling a stylization that maximizes tags (e.g., "volcano") on the output smoke.

Differently from previous image stylization approaches, we do not enforce the matching of content loss to the original unstyled image, and instead, the content loss is used to drive the flow data towards the creation of patterns. Since the smoke is modified by advecting its density towards stylization, we can guarantee that each iteration of the optimization will only slightly modify the original smoke by normalizing the gradients for updating the velocities with the fixed learning rate size (η).

4.2.3 Style Transfer

In addition to semantic transfer, which is designed to use "built-in" features of the pre-trained network *without* any reference image as input, our method allows the incorporation of a given input image style, as shown in Figure 4.13. The style is computed by correlations between different filter responses, where the expectation is taken over the spatial extension of the input image. Hence, in contrast to semantic transfer, we minimize the difference between feature distributions. Given $\hat{\mathcal{F}}^l_k(I)$, which is the flattened one-dimensional version of a 2D filter map at the k^{th} channel, the Gram matrix entry for two channels m and n is

$$G^l_{mn}(I) = \sum_{i}^{H_l \times W_l} \hat{\mathcal{F}}^l_{mi}(I)\, \hat{\mathcal{F}}^l_{ni}(I), \tag{4.6}$$

where i iterates over all pixels of the vectorized filter. Thus, the Gram $G^l(I)$ matrix of a l^{th} layer has dimensions $C_l \times C_l$. The Gram matrix computes the dot product between all filter responses from a layer, storing correspondences of channels denoted by the row and column of an entry. The user-specified parameter $\mathbf{p_s}$ consists of a target image I_s and a set of layers for which the style will be optimized for. Thus, the normalized loss function $\mathcal{L}_s$ for matching styles between an input image and a target style image is

$$\mathcal{L}_s(I, \mathbf{p}_s) = \sum_{l}^{L} \left[\frac{1}{4C_l^2(H_l \times W_l)^2} \sum_{m,n}^{C_l} \left(G^l_{mn}(I) - G^l_{mn}(I_s) \right)^2 \right]. \tag{4.7}$$

Similarly to our semantic transfer, the Gram matrix layer choice in Equation (4.7) controls different abstraction levels of the stylization, as illustrated in Figure 4.6. However, the style transfer does not match features that have spatial correlations relative to the input image, but rather approximates filter response statistics. We further highlight differences between semantic and style transfer in Section 4.4.1.

4.2.4 Time-Coherent Stylization

As densities are updated with the simulation advancement, distinct features can be emphasized by semantic and style transfer losses over different frames. Thus, flickering will occur if time-coherency between frames is not enforced explicitly, as shown in Figure 4.7. Given that the velocities of the original simulation transport densities over time, we use them to align stylization velocities computed independently for different frames. Once these velocities are aligned, we update a single frame stylization velocity field by smoothing subsequent aligned velocities together. Specifically, we define $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{n-1}, \mathbf{u}_n\}$ as the set of simulation velocities computed for the whole simulation duration. The advection function $\mathcal{T}_i^j$ that takes a stylization velocity at the i^{th} frame to the j^{th} frame is

$$\mathcal{T}_i^j(\mathbf{v}_i, \mathbf{U}) = \begin{cases} \mathcal{T}(\dots \mathcal{T}(\mathcal{T}(\mathbf{v}_i, \mathbf{u}_i), \mathbf{u}_{i+1}) \dots, \mathbf{u}_{j-1}), & \text{if } i < j \\ \mathcal{T}(\dots \mathcal{T}(\mathcal{T}(\mathbf{v}_i, -\mathbf{u}_{i-1}), -\mathbf{u}_{i-2}) \dots, -\mathbf{u}_j), & \text{if } i > j\,, \\ \mathbf{v}_i, & \text{if i = j} \end{cases} \tag{4.8}$$

where $\mathcal{T}$ is a function that advects a velocity or a density field for a single time-step. Equation (4.8) is recursive, and aligning a velocity field defined n frames away from a specific frame requires n evaluations of the advection function. A temporally coherent velocity for stylization of frame t is given

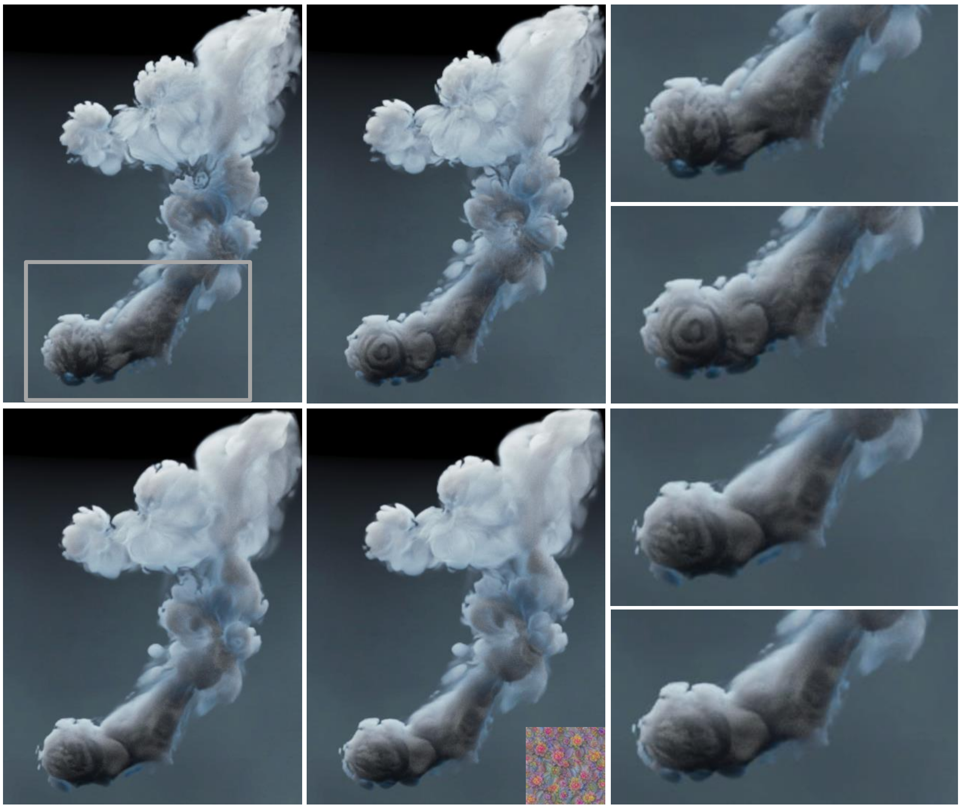


Figure 4.7: *Two subsequent frames of the smoke jet example stylized with a flower motif [gfv, 2015], no time coherence (top) and window size 9 (bottom). For each frame, a close-up view corresponding to the highlighted region is shown on the right. Using our algorithm with a bigger window size ensures that the structures created in one frame are propagated to subsequent stylizations.*

as a linear combination of aligned neighbor velocity fields

$$\mathbf{v}_t^* = \sum_{i=t-w}^{t+w} \omega_i \mathcal{T}_i^t(\mathbf{v}_i, \mathbf{U}), \tag{4.9}$$

where w is the number of neighboring frames evaluated in time, $2w + 1$ is the window size and ω_i is a weighting term. Let $\mathbf{V}_t = \{\mathbf{v}_{t-w}, \mathbf{v}_{t-(w-1)}, \ldots, \mathbf{v}_t, \ldots, \mathbf{v}_{t+(w-1)}, \mathbf{v}_{t+w}\}$ be the window of stylization velocities at time t obtained by the combination of corresponding potential windows $\mathit{\Phi}_t, \mathbf{\Psi}_t$ defined in a range from $t - w$ to $t + w$. The time-coherent multi-view stylization optimization is

$$\hat{\mathit{\Phi}}_t, \hat{\mathbf{\Psi}}_t = \operatorname*{arg\,min}_{\mathit{\Phi}_t, \mathbf{\Psi}_t} \sum_{i=t-w}^{t+w} \sum_{\theta \in \Theta} \mathcal{L}(\mathcal{R}_\theta(\mathcal{T}(d_i, \mathbf{v}_i^*)), \mathbf{p}_c, \mathbf{p}_s). \tag{4.10}$$

In practice, evaluating directly Equation (4.10) becomes infeasible as the number of neighbors increases. The memory used by the automatic differentiation procedure to compute derivatives quickly grows as the window size increases. Thus, we approximate the solution of Equation (4.10) by first evaluating Equation (4.4) to find a set of stylization velocities computed for a single frame. Then, we merge the velocities per-frame individually using Equation (4.9). This is performed iteratively for all simulation frames of a sequence, and the multi-view time-coherent process is summarized in Algorithm (2).

Algorithm 2: *Multi-View Time-Coherent Smoke Stylization*

```
while $i < n_{iter}$ do
  while $t < n_{frames}$ do {// single frame stylization}
    Compute density by $d_t^* = \mathcal{T}(d_t, \mathbf{v}_t^*)$
    for $\theta \in \Theta$ do
      Render $I_t^\theta = \mathcal{R}_\theta(d_t^*)$ with angle $\theta$
      Obtain $\nabla\Phi_t^\theta, \nabla\mathbf{\Psi}_t^\theta$ from $\mathcal{L}(I_t^\theta, \mathbf{p}_c, \mathbf{p}_s)$
    end for
    Merge gradients from views $\nabla\Phi_t^\theta, \nabla\mathbf{\Psi}_t^\theta$ to obtain $\nabla\Phi_t^*, \nabla\mathbf{\Psi}_t^*$
  end while
  while $t < n_{frames}$ do {// temporal alignment}
    for $t_w = t - w, t_w < t + w$ do
      Align gradients $\nabla\Phi_t^*, \nabla\mathbf{\Psi}_t^*$ to obtain $\nabla\Phi_t, \nabla\mathbf{\Psi}_t$ using Equation (4.9)
    end for
    $\Phi_t = \Phi_t + \eta\nabla\Phi_t, \mathbf{\Psi}_t = \mathbf{\Psi}_t + \eta\nabla\mathbf{\Psi}_t$
    $\mathbf{v}_t^* = \lambda\nabla\Phi_t + (1 - \lambda)\nabla \times \mathbf{\Psi_t}$
  end while
end while
```

4.3 Differentiable Smoke Renderer

Similar to the flat shading approach proposed by Liu et al. [2018b] for stylizing meshes, our smoke renderer is lightweight. The optimization of Equation (4.4) heavily relies on rendered density representations, and an overly sophisticated volumetric renderer compromises efficiency. Our renderer outputs grayscale images, in which pixel intensity values will correspond to density occupancy data. Thus, modelling smoke self-shadowing would map shadowed regions to empty voxels on the rendered image. Nevertheless, our results show that meaningful correspondences between the stylization velocities and density fields can be computed on representations that do not match perfectly the ones produced by the final rendered image.

The smoke stylization optimization usually performs many iterations, computing derivatives of the loss function (Equation (4.4)) with respect to the ve-

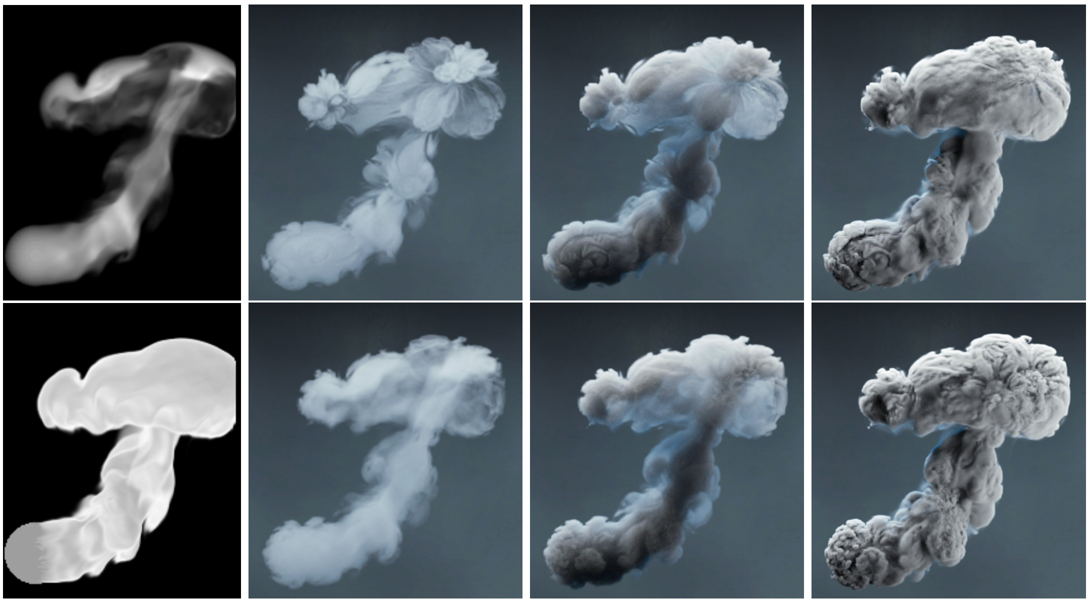


Figure 4.8: *The value of γ controls how the smoke density is stylized. Smoke images (left) produced by our renderer with $\gamma = 0.01$ (top) and $\gamma = 1$ (bottom). The final smoke renderer is configured with varying thickness, highlighting how the stylization gets transferred for different smoke appearances.*

locity field by automatic differentiation. Therefore, the volumetric rendering requires efficiency. Our lightweight differentiable rendering algorithm only incorporates a single directional light traced directly from the pixel rendered from an orthographic camera. We measure how much of this single light ray gets transmitted through the inhomogeneous participating media, which is described by [Fong et al., 2017], to compute the transmittance and the image pixel grayscale value as

$$\begin{aligned} \tau(\mathbf{x},\mathbf{r}) &= e^{-\gamma \int_{\mathbf{x}}^{\mathbf{r}_{max}} d(\mathbf{r})\, dr} \\ I_{ij} &= \int_0^{\mathbf{r}_{max}} d(\mathbf{x})\, \tau(\mathbf{x},\mathbf{r})\, dx, \end{aligned} \tag{4.11}$$

where $\mathbf{r}_{ij}$ is a vector traced from pixel ij into the normal direction of an orthographic camera, $d(\mathbf{r})$ is the density value, γ is a transmittance absorption factor, and r_{max} is the maximum length of the traced ray. The value computed at each image pixel is the integral of the transmittance multiplied by the density values, mapping empty and full smoke voxels to 0 and 1, respectively. We additionally multiply the transmittance and densities along the integration ray since it generates richer features for thicker smoke scenarios. Comparisons between this approach against simply integrating the transmittance along the view-ray are discussed in Section 4.4.2.

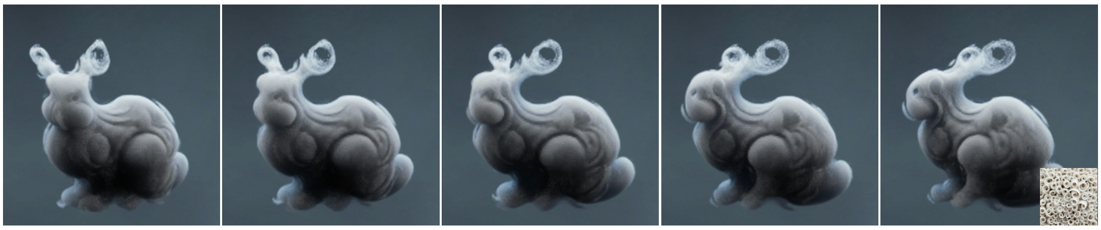


Figure 4.9: *Stanford Bunny shaped smoke stylized with spiral patterns [spi, 2015] for multiple views ([-30°,30°], every 15 degrees). Our method focuses the instantiation of patterns on smoke regions that share similarities with the target motif. Additionally, augmented flow structures change smoothly when the camera moves around the object.*

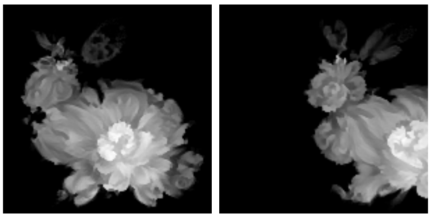
The smoke density is linearly mapped to extinction using the scaling factor γ, which determines how quickly light gets absorbed by the smoke. Figure 4.8 shows that minimizing the discrepancy between the final rendered smoke and the representation in which it is optimized is important. Setting low transmittance constants in the stylization renderer will result in more aggressive smoke modifications towards the normal view direction, while high transmittance will over-constrain the stylization velocity field to the smoke surface that is closer to the camera. Figure 4.8 shows the effect of varying γ values with the final rendered smoke thickness: low γ values will produce better results for shallow smoke appearances, while higher γ values will more efficiently stylize thicker smoke.

4.3.1 Camera Design Specifications

Participating media naturally incorporates transparency, and a single-view stylization update will be propagated inside the volumetric smoke even though the rendered image is two dimensional. Therefore it is not necessary to uniformly cover every viewpoint of the smoke with equal probability as in [Liu et al., 2018b]. Given a pre-defined camera path, we use Poisson sampling [Bridson, 2007] around a small area of its trajectory (Figure 4.11, left) to avoid bias that would be introduced by a fixed set of viewpoints.

Since feature maps obtained from 2D views of the camera are used, we specify that the image rendered by the camera is invariant to zooming, panning and rolling. This means that if the camera is moving (as in Figure 4.11, left), our renderer automatically centers the smoke representation in the frame, only responding to variations of the viewing angle. These invariances ensure that filter map activations remain constant as long as no new voxels are shown in the rendered image; rotations, however, have to be accounted for. Thus, our renderer camera position is parameterized by the polar coordinates tuple $\theta = (\theta_1, \theta_2)$, while the camera always points to a fixed point inside the smoke. Note that this simplification is only possible since we are adopting an orthogonal camera, and a perspective projection might reveal new voxels with translational movement.

The inset image shows how enhanced features (e.g., patterns at the bunny face) vary due to translations in the image space, in which the stylization should remain constant.



In order to evaluate Equation (4.11) for multiple perspectives, we need to integrate smoke voxels along the camera view direction. Implementing a classic ray-marching sampling along an arbitrary ray direction is challenging in Deep Learning frameworks, which are usually optimized for tensor operations. Thus, we adopted the spatial transformer network (STN) of Jaderberg et al. [2015]. The STN instances a rotated 3D domain with the same dimensionality as the original one that is aligned with the camera view as illustrated in Figure 4.11 (right). This allows us to evaluate samples by evoking simple built-in features that implement voxel summations along the view direction of each pixels' ray. These specifications make our rendering algorithm efficient, accounting for about 30% of time taken for processing a batch (see Table (4.2)).

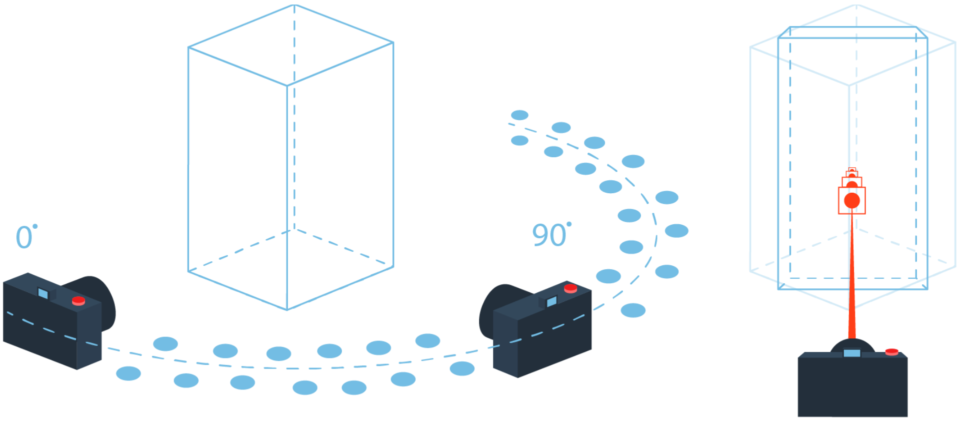


Figure 4.11: *Multi-View camera configuration. We sample a camera path with Poisson sampling, which prevents smoothing of density details between pre-defined viewpoints (left). The volumetric smoke grid is aligned with the camera viewpoint to facilitate light ray integration (right).*

4.4 Results

We demonstrate that our approach can reliably transfer various styles from images onto volumetric flow data, with automatic semantic instantiation of features and artistic style transfer. All our stylization examples employed a mask with soft edges (Figure 4.16) that is extracted from the original smoke data. The mask is applied to the potential field, and it restricts modifications to be close to the original smoke border while enhancing temporal accuracy near smoke boundaries (Section 4.4.2). The advection operator $\mathcal{T}$ is implemented by the MacCormack method [Selle et al., 2008]. We refer the reader to the supplemental video for the corresponding animations.

Equations 4.5 and 4.7 are optimized by stochastic gradient descent, with the gradients computed by backpropagation on GoogleNet [Szegedy et al., 2015]. Although we use automatic differentiation, analytic differentiation would allow us to fit even bigger simulation examples [Liu et al., 2018b]. We modified the original stride size of GoogleNet's first layer from two to one to remove checkerboard patterns that occur when the kernel size is not divisible by the CNN's stride size [Odena et al., 2017]. We use a fixed learning rate and apply multi-scale stylization and Laplacian pyramid gradient normalization techniques [Mordvintsev, 2016] for boosting lower frequencies. Parameters and performance values for all examples are summarized in Table (4.2). Our implementation uses *tensorflow* [Abadi et al., 2016] evaluated on a TITAN Xp GPU (12GB). The input simulations have been computed with different solvers. We used *mantaflow* [Thuerey and Pfaff, 2018] for the smokejet and bunny examples in Figure 4.12 and Figure 4.13, Houdini for computing the volcano in Figure 4.1, and a dataset from Sato et al. [2018a] in Figure 4.14.

4.4.1 Semantic and Style Transfers

To demonstrate how our method performs under distinct style and semantic transfers, we designed two instances of buoyancy-driven smoke: a smokejet with a sphere-shaped source and an initial horizontal velocity, and a smoke initialized with the Stanford bunny shape (Figure 4.12, left). For all examples shown in Figures 4.12 and 4.13 we used 20 iterations for each scale with a learning rate of 0.002, 3 Laplacian subdivisions and 9 camera views for a single frame Poisson sampled around the original view with elevation (θ_1) and azimuth (θ_2) ranges spanning $[-5°, 5°]$ and $[-10°, 10°]$ respectively. The stylized examples show that our method is able to augment the original flow structures of the smoke, generating a wide set of artistic and natural 3D effects.

Examples in Figure 4.12 demonstrate results obtained by applying the semantic style transfer loss from Equation (4.5). All the feature maps are from the GoogleNet [Szegedy et al., 2015] architecture; the captions indicate which layer of the CNN is used for the optimization. Activating different layers of the network will yield stylization results which will vary in the complexity of instantiated structures. In the two first examples we used filters closer to initial layers, which depict simpler patterns that occur at lower levels of abstraction. These patterns are used by higher levels to composite more complex structures. As the layers become deeper, the network is able to produce more intricate motifs, such as structures similar to flowers, fur, or ribbons.

Examples shown in Figure 4.13 apply the style loss of Equation (4.7). To demonstrate the flexibility of our approach, we used three different image categories for testing the style transfer loss: photorealistic (first and second columns), artistic (third and fourth columns) and patterns (fifth and sixth columns). For all these examples, a mix of convolution layers from different levels of the CNN is used, similarly to [Gatys et al., 2016a]. The representations of the layers employed on the style transfer are depicted below each input style image. Figure 4.2 shows distinct frames of the bunny-shaped rising smoke stylized with the volcano and spiral input images from Figure 4.13. In Figure 4.14 we compare our results with the example-based turbulence transfer method of Sato et al. [2018a]. The style of the single frame rendering (Figure 4.14, middle) of their method output is transferred by our approach to an input coarse simulation. The results show that our approach is able to generate similarly detailed flow structures with only a given reference image.

Besides individually transferring semantics and styles, our method is also flexible to allow the combination of these techniques. An example in the accompanying video demonstrates the semantic instantiation of cloud motifs merged with the style of a fire texture, while a similar stylization shows ribbon patterns combined with Starry Night painting style. Figure 4.1 shows a low-resolution volcanic setup in which a combination of cloud motifs and a volcano texture was used to create turbulent details. The thick smoke produced by volcanic ashes poses challenges to our renderer, since it quickly saturates transmittance values. Nevertheless, our renderer is able to create structures that consistently correlate with the input smoke.

Our method is also able to control the amount of smoke dissipation by the decomposition of the stylization velocity field into its incompressible and ir-

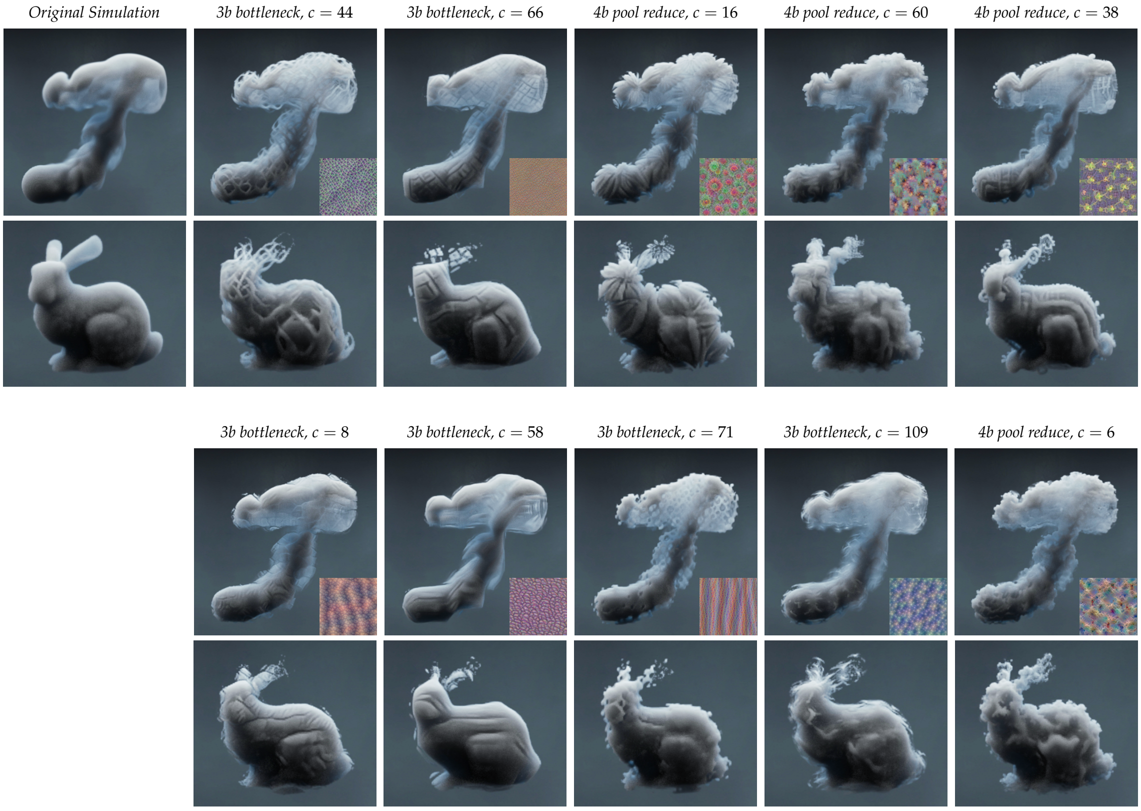


Figure 4.12: *Semantic transfer applied to a smokejet and bunny simulations (leftmost column). Images on the same column are stylized with the feature map depicted on the right corner [gfv, 2015]. The examples for semantic transfer depict different levels of abstraction, showing patterns that occur at shallow levels of the network (first row, first two columns, second row, first four columns) and intricate motifs that are represented at deeper levels (first row, last three columns, second row, last column).*

rotational parts. Figure 4.5 and examples in the supplemental video show that different artistic patterns are created depending on the constraints imposed on the velocity field.

Although our method is based on 2D representations of the smoke data, it can reliably cover multiple viewing directions without introducing bias towards certain views. This is allowed by the Poisson sampling of positions along the camera trajectory. Figure 4.9 shows the bunny smoke example stylized with a spiral pattern from different viewpoints. Transferred structures change smoothly when the camera moves around the object. An example in the accompanying video compares results of the smoke bunny example stylized with single view and multiple camera views. The multiple camera

Table 4.2: *Parameters and performance statistics. We used a constant multi-scaling factor of 1.8, and the input size is firstly down-sampled to* 61 × 92 × 61 *and up-scaled to* 111 × 166 × 111 *and* 200 × 300 × 200*. Computation time per frame includes all input scales.*

	Simulation		Learning	Extinction	Multi	# Target	Computation
Scene	Resolution	# Frames	Rate	Factor α	Scale	Layers	Time per Frame (m)
Semantic Transfer (Fig. 4.12)	200 × 300 × 200	120	0.002	0.1	3	1	13.47
Style Transfer (Fig. 4.13)	200 × 300 × 200	120	0.002	0.1	2	3	12.68
Volcano (Fig. 4.1)	200 × 300 × 200	140	0.003	10	2	2	12.97
Sato et al. [2018a] (Fig. 4.14)	192 × 256 × 192	140	0.005	5	2	3	11.97

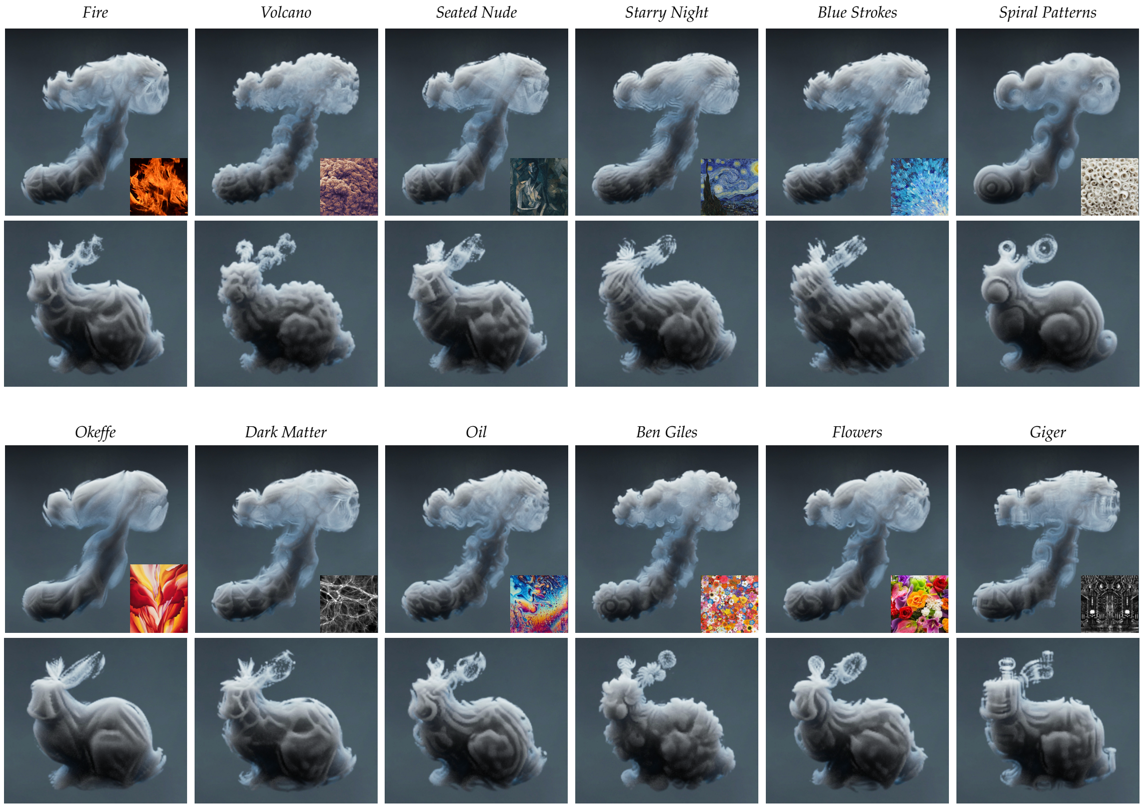


Figure 4.13: *Style transfer applied to a smokejet and bunny simulations. We used photorealistic (first two columns), artistic (middle two columns) and pattern-based (last two columns) input images [sti, 2018] as input to the stylization algorithm. Fire examplar ©Bunzellisa via pixabay.*

views are Poisson sampled with 180 degrees angle range around the $y-$axis. The multi view approach shows more salient 3D structures when compared with the single view stylization.

Lastly, Figure 4.7 shows the impact of different time coherency window sizes (Equation (4.9)). We compare a window size of 1 (frame-based stylization) with a window size of 9 (used in all other examples) for multiple subsequent frames. Features heavily flicker with a small window size, while augmented structures change smoothly with larger windows. These results are better visualized in the accompanying video, in which we also included a comparison with a window size of 5.

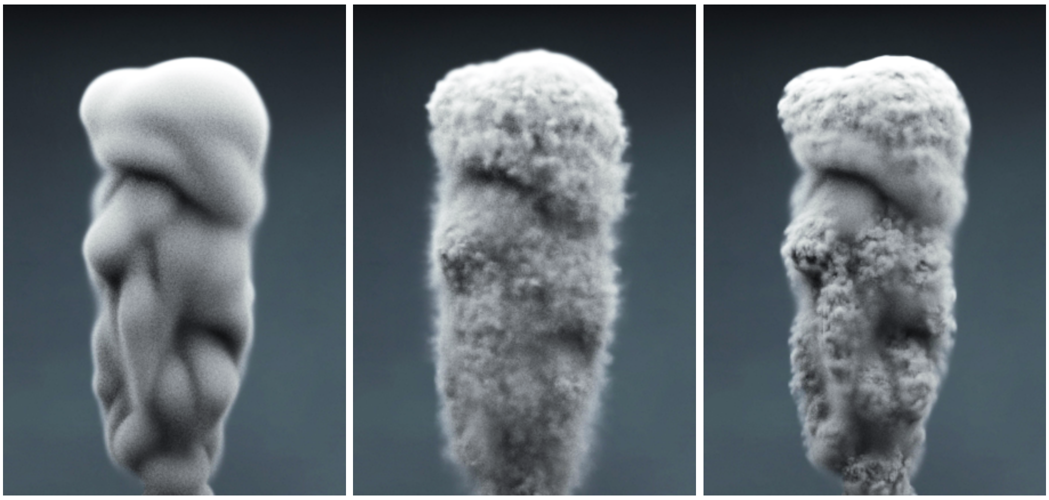


Figure 4.14: *Style Transfer comparison. From left to right: input density field of Sato et al. [2018a], the result of applying the method of Sato et al., and our style transfer result using the middle image as stylization input.*

4.4.2 Discussion

Differentiability Note that all operators including advection and rendering require differentiability with respect to the velocities, so efficient gradient-based optimization methods can be employed. Traditional NST works by differentiating the loss of a classification network with respect to the image input, computing gradients of filter responses to image variations. Since classification networks convolve images to create filter responses, these filters are assumed to smoothly change with respect to image variations, and thus NST works without differentiability issues. This is the same for our work, however we additionally require that both the transport towards stylization and the smoke rendering to be differentiable. The rendering scheme denoted by Equation (4.11) is clearly differentiable. The MacCormack advection uses the Semi-Lagrangian method as its building blocks to correct error estimations. The correction is differentiable and the Semi-Lagrangian algorithm works by sampling densities in previous positions. Thus, two components need to be considered for differentiation to work: estimation of particle trajectories and density sampling. The estimation of the particle trajectories is a linear ODE, and thus differentiable. Densities are estimated by grid sampling, and as shown in [Jaderberg et al., 2015] this is also differentiable when using linear interpolation kernels.

Performance and memory limitations Table (4.2) shows the average time for stylizing a single frame of different simulation resolutions, with grids up to 200 × 300 × 200. Performance was not the focus of this work, and as shown by extensive follow-up works to image-based NST [Ulyanov et al., 2016], we believe that real-time stylizations can be obtained by training networks to directly output stylized results. For higher resolutions, the limiting factor is the single GPU memory used for computing the backpropagation with *tensorflow* automatic differentiation. As in [Liu et al., 2018b], the memory limitation could be greatly reduced by using analytic differentiation.

Temporal Coherency and Boundary conditions Features are instantiated by evaluating smoke representations independently for each frame. Our temporal coherency algorithm aligns and blends the creation of those features; however, due to the nature of the underlying physical phenomena, smoke structures might appear and disappear as the simulation advances. This might induce abrupt changes in the stylized smoke, specially when considering smoke edges. We did not post-process our results in order to have a fair evaluation, but this effect can be controlled by blending the results with the original smoke simulation or by a more aggressive masking scheme. Regarding boundary conditions, the final stylized smoke can only slightly penetrate objects inside the simulation, since it starts from a density field configuration that is already boundary respecting. We provide an example of stylization on smokejet simulation with a sphere shape obstacle to see how our method works on the presence of boundaries. Note that the mask is not applied in this case, since the masking would guarantee zero penetration on boundaries. Figure 4.15 shows density field visualizations of the base simulation, stylized density field, simulation velocity fields and stylized velocity fields, from top to bottom. Note that since the original simulation velocity field is already boundary-respecting, thus our method only yields slight penetrations on the obstacle boundary.

Soft Masking Examples in this thesis apply a soft mask (Figure 4.16, right) to the stylization velocity field, in order to conform the stylization output to the original smoke silhouette. Figure 4.16 compares results obtained without employing the soft mask for irrotational, mixed and incompressible examples. Not applying the mask causes the smoke to spread, especially in the case of incompressible velocity fields.

Control Parameters Figure 4.17 shows the amount of smoke dissipation over time by the decomposition of the stylization velocity field into its incompressible and irrotational parts. As expected, a streamfunction-based

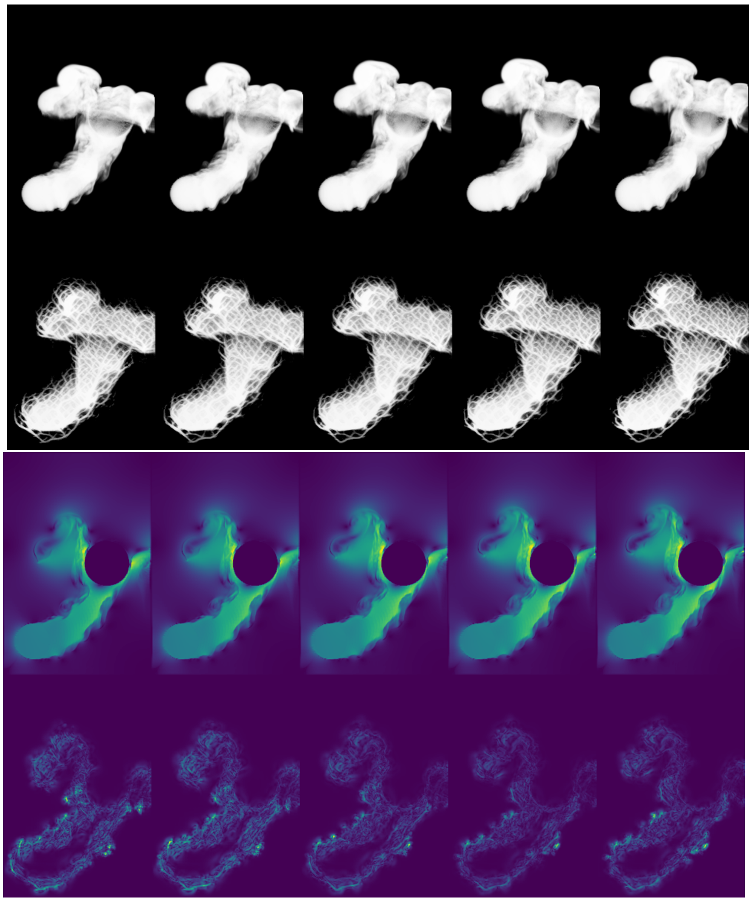


Figure 4.15: *Stylization on smokejet simulation with a sphere shape obstacle. First row shows base simulation, second row shows stylized density fields, third row represents the middle slice view of magnitude of base simulation velocity fields, while fourth row shows those of stylization velocity fields. Note that no soft mask is used.*

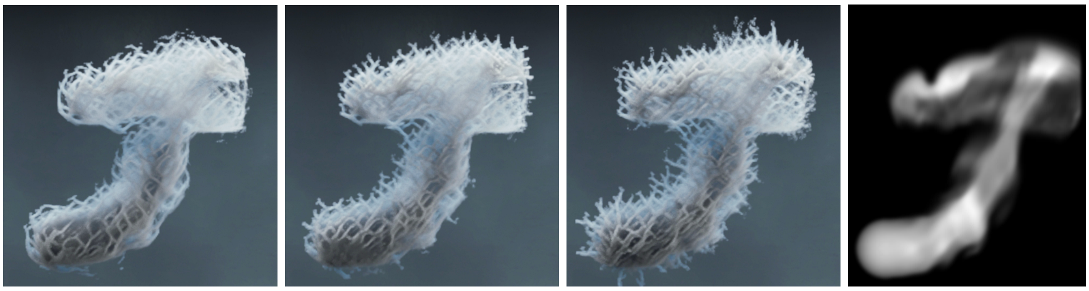


Figure 4.16: *Results from semantic transfer of a net structure without soft mask (rightmost). Irrotational, mixed and incompressible velocity fields, from left to right.*

(incompressible) velocity stylization shows less dissipation than those of potential field based (irrotational) velocity stylization. Conservation errors due numerical integration on the advection algorithm, although, are still present. This causes the streamfunction-based velocity field to not exactly conserve the original amount of smoke.

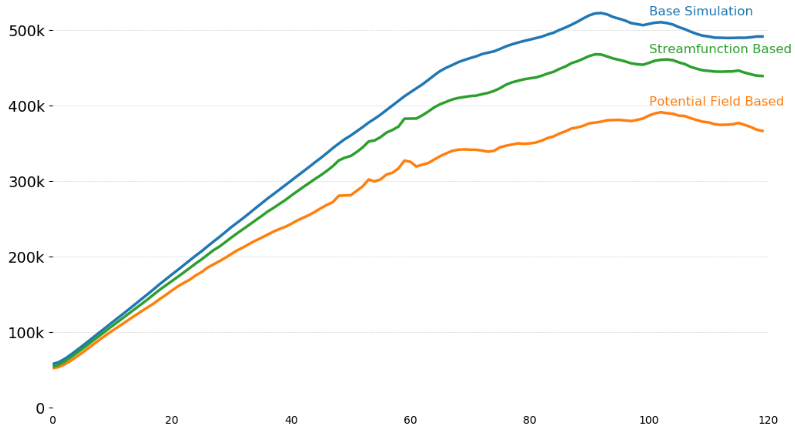


Figure 4.17: *Density amount plot comparison for incompressible and irrotational velocity fields. Using a streamfunction-based (incompressible) velocity fields reduces the loss of density amount compared to an irrotational approach.*

Figure 4.18 shows the effects of varying the number of iterations and the learning rate size. The learning rate affects structures significantly, higher learning rates result in more pronounced details. This is also visible in Figure 4.19 where different learning rates for the Starry Night Style transfer were used.

Analysis on Differentiable Rendering Methods We compared a simpler alternative rendering method to the Equation (4.11). The image is calculated by simply integrating the transmittance along the viewing ray as

$$I_{ij} = 1 - e^{-\alpha \int_0^{r_{max}} d(\mathbf{r}_{ij})\, dr}. \tag{4.12}$$

Figure 4.20 shows the difference between the rendering method proposed on this thesis and the one in Equation (4.12). We vary the transmittance absorption factor (γ) for the stylization of the bunny example (left to right); top image sequence shows the differentiable rendering method, while bottom one shows the alternative rendering computed from Equation (4.12). While low transmittance values (leftmost) produce similar renderings and stylizations for both approaches, using Equation (4.12) quickly saturates the

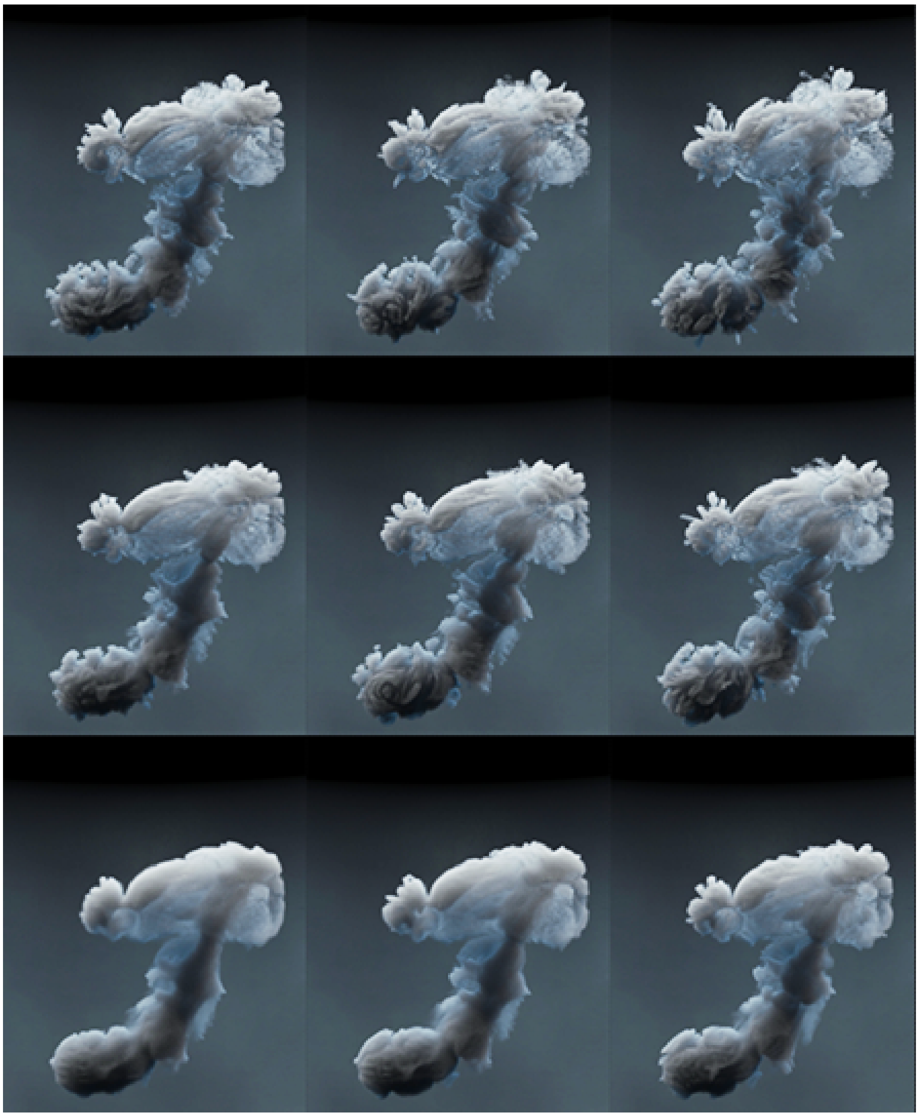


Figure 4.18: *Influence of iteration number and learning rate. From left to right: 5, 10, and 20 iterations. From top to bottom: learning rate of 0.001, 0.0005, 0.0001.*

image as γ increases. The differentiable rendering method, however, creates structures that are presented even in examples with higher transmittance absorption factors (rightmost).

2D Examples We compare the value-based and transport-based density optimization for a 2D smoke simulation in Figure 4.21. The value-based method shows sharper details, but introduces ghosting artifacts in temporal sequences as spurious density sources and sinks are added. Our transport-based approach leads to temporally smoother stylizations as the total vol-

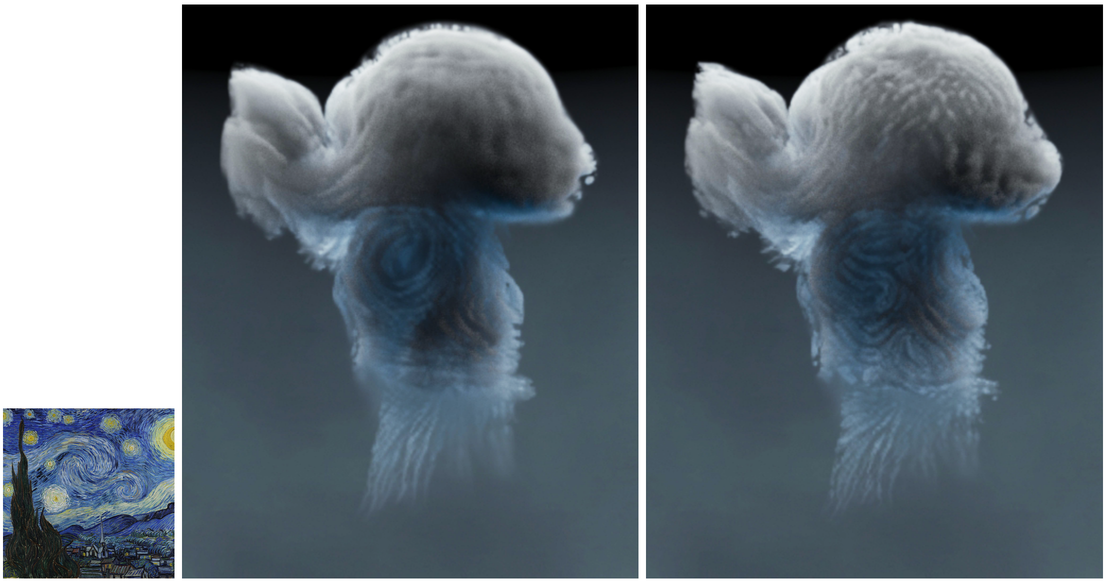


Figure 4.19: *Influence of using different learning rates for the Starry Night style transfer example. A higher learning rate (right, 5× higher) results in more pronounced structures than when using lower learning rates (left), but also more noisy results.*

ume is conserved. We also compare the temporal coherency with different window sizes for a 2D smoke simulation in Figure 4.22. The images are cropped from the top right part of the stylized image sequence. The top row shows the results using a window size of 1, and the bottom row is generated with a window size of 9. The larger the window size, the more overall structure is conserved (highlighted by the color-coded circles).

Figure 4.23 shows different 2D semantic and style transfer examples. We also provide comparisons about different abstraction levels of style features for 2D simulations (Figure 4.24). We can control low (left), medium (middle) and high (right) levels of features. Figure 4.25 shows how changing the resolution of the input fluid simulation affects the stylization. Higher resolution simulations (right) lead to sharper structures than coarse simulations (left).

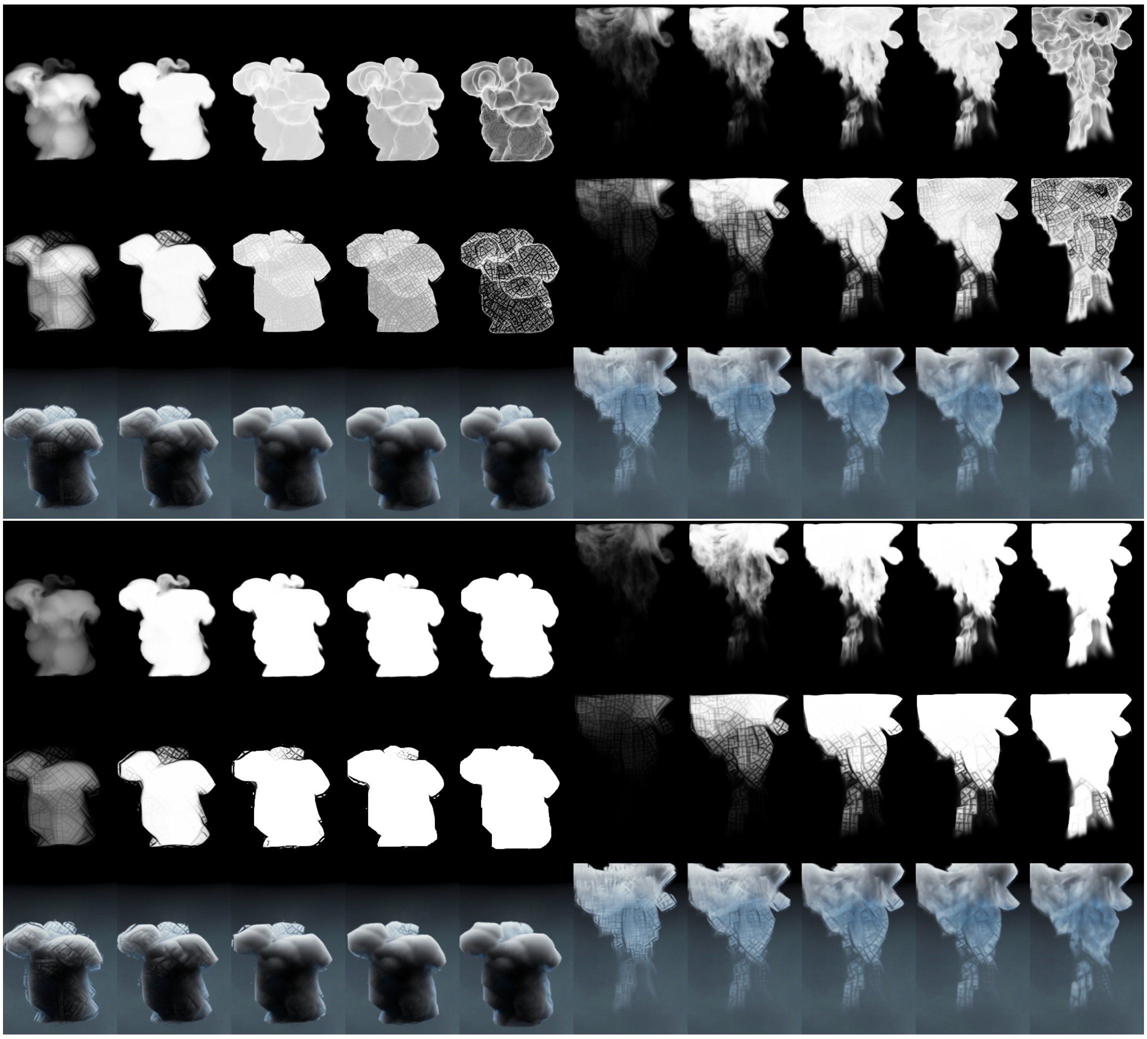


Figure 4.20: *Our differentiable rendering method (top) versus the one with Equation (4.12) (bottom).*

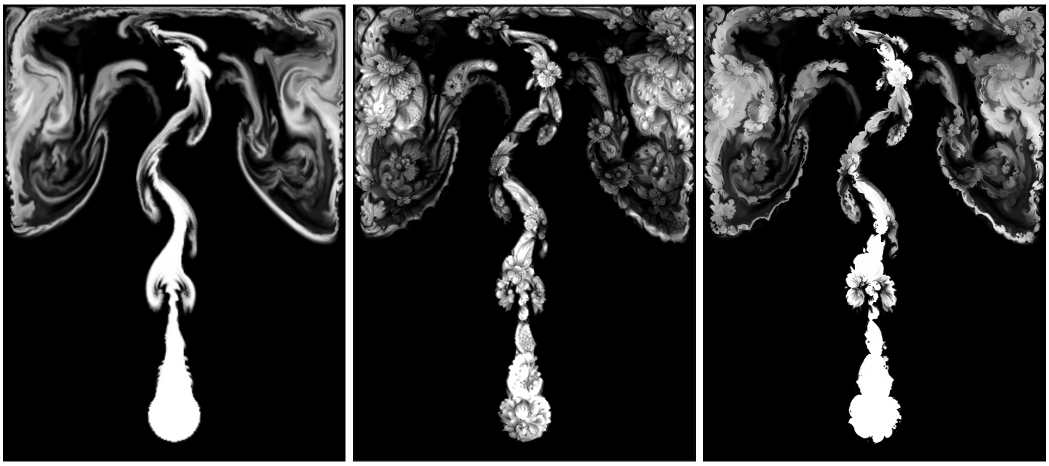


Figure 4.21: *Value-based density optimization (middle) versus transport-based density optimization (right). The input smoke simulation is shown on the left.*

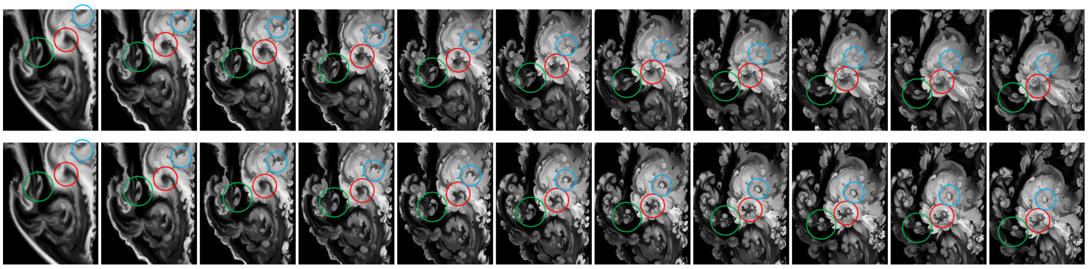


Figure 4.22: *Comparison of temporal coherency using different window sizes of 1 and 9.*

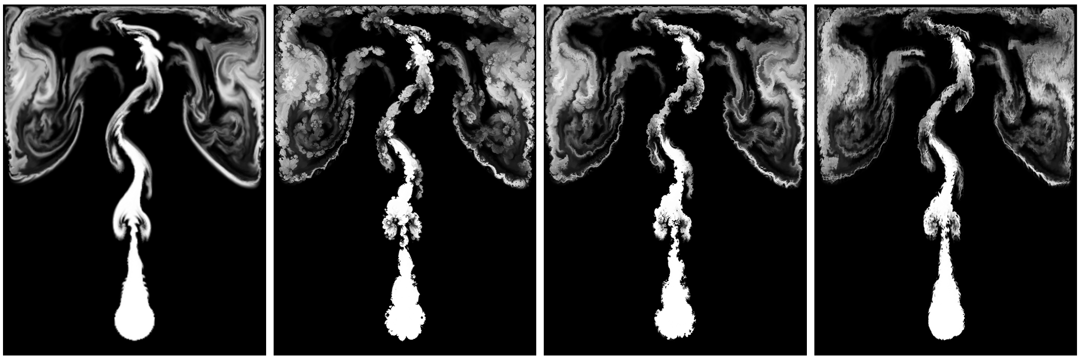


Figure 4.23: *From left to right: coarse input simulation, flower, volcano, and fire stylizations.*

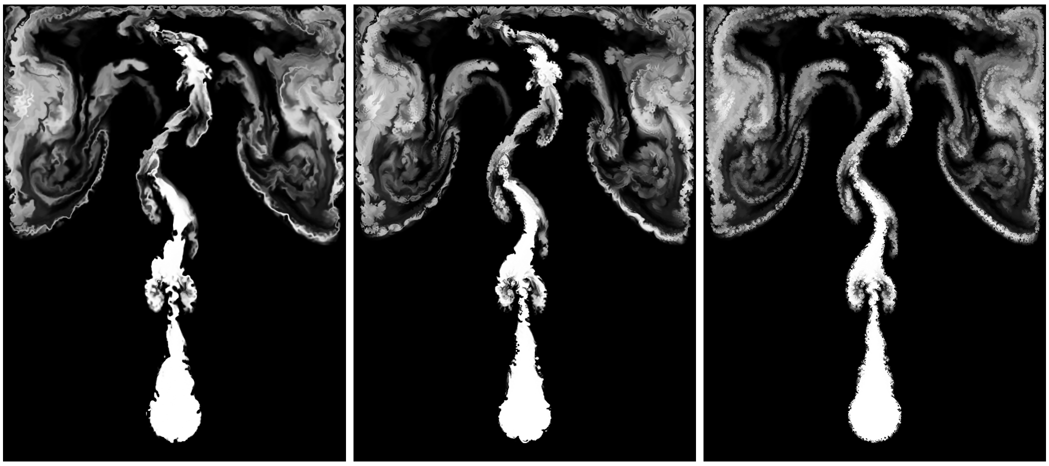


Figure 4.24: *Low (left), medium (middle) and high (right) levels of style features.*

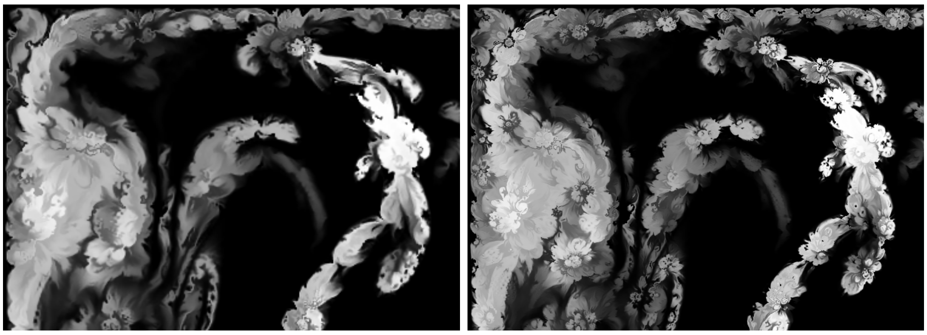


Figure 4.25: *Stylization applied to a low-resolution (left) and high-resolution (right) fluid simulation. More detailed structures are synthesized with higher resolutions.*

CHAPTER 5

Lagrangian Neural Artistic Control of Fluid Simulations

This chapter is based on the following publication by [Kim et al., 2020]:

B. Kim, V. C. Azevedo, M. Gross, B. Solenthaler, **Lagrangian Neural Style Transfer for Fluids**, *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2020)*, 39(4), Jul. 2020. (selected for the video trailer and back cover of the proceedings)

The supplemental video can be found here: `https://www.youtube.com/watch?v=WPmUsIVf3-4`.

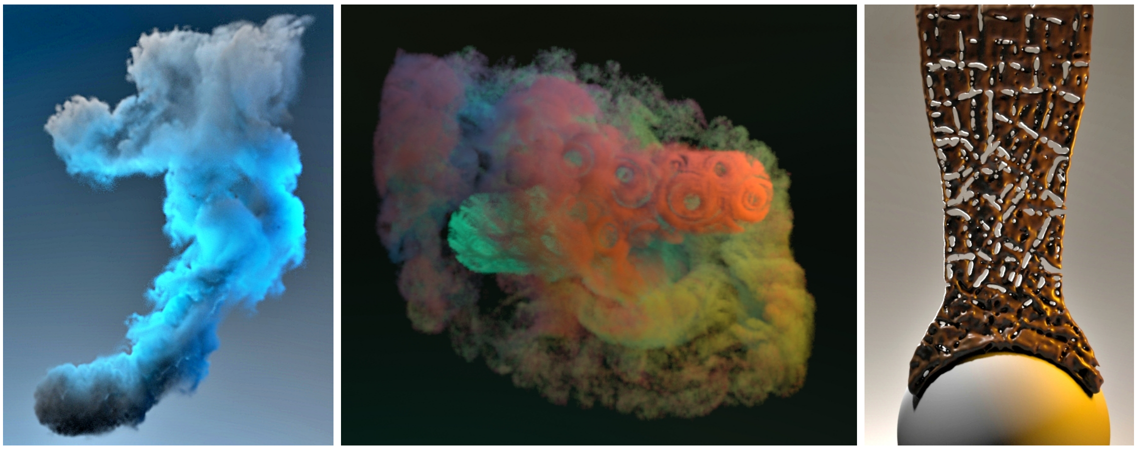


Figure 5.1: *Our Lagrangian neural style transfer enables novel artistic manipulations, such as time-coherent stylization of smoke, multiple fluids and liquids.*

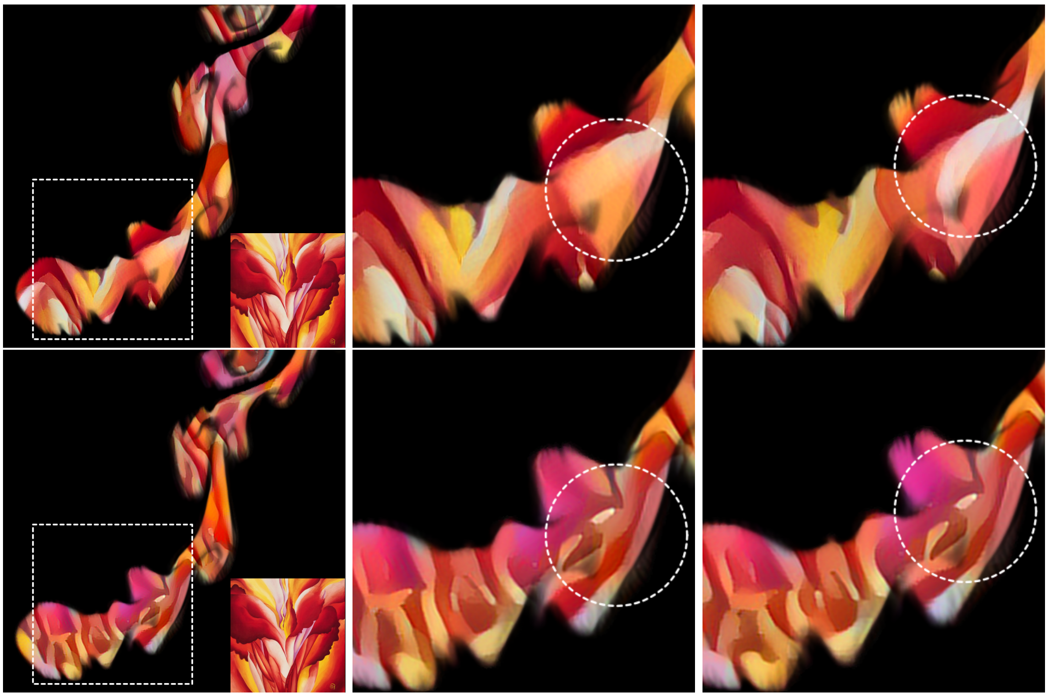


Figure 5.2: *Neural color stylization [Christen et al., 2020] using the input* Red Canna *applied to a smoke scene with TNST (top) and LNST (bottom). The close-up views (dashed box, frames 60 and 66) reveal that LNST is more time-coherent than TNST (dashed circle).*

5.1 Overview

In the previous Chapter 4, we show that our Transport-based Neural Style Transfer (TNST) [Kim et al., 2019a] takes flow appearance and motion control to a new level: arbitrary styles and semantic structures given by 2D input images are automatically transferred to 3D smoke simulations. The achieved effects range from natural turbulent structures to complex artistic patterns and intricate motifs. The method extends traditional image-based Neural Style Transfer [Gatys et al., 2016b] by reformulating it as a transport-based optimization. Thus, TNST is physically inspired, as it computes the density transport from a source input smoke to a desired target configuration, allowing control over the amount of dissipated smoke during the stylization process. However, TNST faces challenges when dealing with time coherency due to its grid-based discretization. The velocity field computed for the stylization is performed independently for each time step, and the individually computed velocities are recursively aligned for a given window size. Large window sizes are required, rendering the recursive computation expensive while still accumulating inaccuracies in the alignment that can

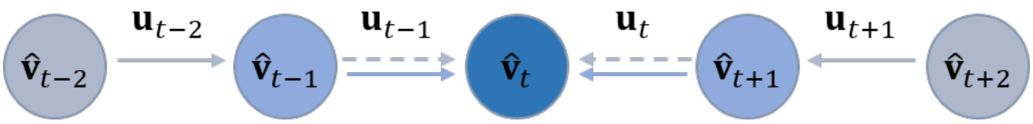


Figure 5.3: *Recursive temporal alignment in TNST. For a window size w, $(w^2-1)/4$ recursive temporal alignment steps are performed for each stylization velocity $\hat{\mathbf{v}}$. Colors indicate the distance to frame t, and arrows refer to advection steps (with recursive steps shown as dashed lines).*

manifest as discontinuities. Moreover, transport-based style transfer is only able to advect density values that are presented in the original simulation, and therefore it does not inherently support color information or stylizations that undergo heavy structural changes.

Thus, in this Chapter 5, we reformulate Neural Style Transfer in a Lagrangian setting (see Figure 5.4), demonstrating its superior properties compared to its Eulerian counterpart. In our Lagrangian formulation, we optimize per-particle attributes such as positions, densities and color. This intrinsically ensures better temporal consistency as shown for example in Figure 5.2, eliminating the need for the expensive recursive alignment of stylization velocity fields. The Lagrangian approach reduces the computational cost to enforce time coherency, increasing the speed of results from one day to a single hour. The Lagrangian Style transfer framework is completely oblivious to the underlying fluid solver type. Since the loss function is based on filter activations from pre-trained classification networks, we transfer the information back and forth from particles to the grids, where loss functions and attributes can be jointly updated. We propose regularization strategies that help to conserve the mass of the underlying simulations, avoiding oversampling of stylization particles. Our results demonstrate novel artistic manipulations, such as stylization of liquids, color stylization, stylization of multiple fluids, and time-varying stylization.

5.2 Lagrangian Neural Style Transfer

In the previous Chapter 4, we found that extending the single frame stylization in a time-coherent fashion is expensive and inaccurate when computed in an Eulerian framework. TNST aligns stylization velocities by recursively advecting them with the simulation velocities for a given window size as shown in Figure 5.3. The recursive nature renders this computation inefficient time- and memory-wise, especially when large window sizes are employed to enable smooth transitions between consecutive frames. Due to

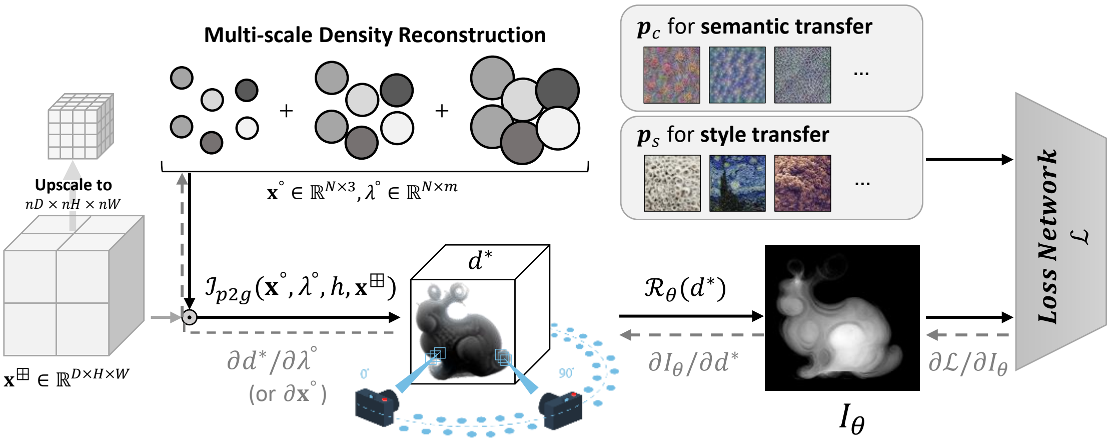

Multi-scale Density Reconstruction
+
+
$\mathbf{x}^{\circ} \in \mathbb{R}^{N\times 3}, \lambda^{\circ} \in \mathbb{R}^{N\times m}$
p_c for semantic transfer
...
p_s for style transfer
...
Loss Network
$\mathcal{L}$
Upscale to
$nD \times nH \times nW$
$\mathbf{x}^{\boxplus} \in \mathbb{R}^{D\times H\times W}$
$\mathcal{I}_{p2g}(\mathbf{x}^{\circ}, \lambda^{\circ}, h, \mathbf{x}^{\boxplus})$
d^*
$\partial d^*/\partial \lambda^{\circ}$
(or $\partial \mathbf{x}^{\circ}$)
0°
90°
$\mathcal{R}_\theta(d^*)$
$\partial I_\theta/\partial d^*$
I_θ
$\partial \mathcal{L}/\partial I_\theta$

Figure 5.4: *Overview of our LNST method. We optimize particle positions $\mathbf{x}^{\circ}$ and attributes λ° to stylize a given density field d^*. We transfer information from particles to the grid with the splatting operation $\mathcal{I}_{p2g}$, and jointly update loss functions and attributes. The black arrows show the direction of the feed-forward pass to the loss network L, and the gray arrows indicate back-propagation for computing gradients. For grid-based simulation inputs, we sample and re-simulate particles in a multi-scale manner (Algorithm (3)).*

the large memory requirement, this operation often has to be computed on the CPU, which generates additional overhead by the use of expensive data transfer operations.

In contrast to its Eulerian counterpart, the Lagrangian representation uses particles that carry quantities such as the position, density and color value. Neural style transfer methods compute loss functions based on filter activations from pre-trained classification networks, which are trained on image datasets. Thus, we have to transfer the information back and forth from particles to the grids, where loss functions and attributes can be jointly updated. We take inspiration from hybrid Lagrangian-Eulerian fluid simulation pipelines that use grid-to-particle $\mathcal{I}_{g2p}$ and particle-to-grid $\mathcal{I}_{p2g}$ transfers as

$$\lambda^{\circ} = \mathcal{I}_{g2p}(\mathbf{x}^{\circ}, \lambda^{\boxplus}) \quad \text{and} \quad \lambda^{\boxplus} = \mathcal{I}_{p2g}(\mathbf{x}^{\circ}, \lambda^{\circ}, h, \mathbf{x}^{\boxplus}), \tag{5.1}$$

where λ° and $\lambda^{\boxplus}$ are attributes defined on the particle and grid, respectively, $\mathbf{x}^{\circ}$ refers to all particle positions, $\mathbf{x}^{\boxplus}$ are grid nodes to which values are transferred, and h is the support size of the particle-to-grid transfer.

Our grid-to-particle transfer employs a regular grid cubic interpolant, while the particle-to-grid transfer uses standard radial basis functions. Regular Cartesian grids facilitate finding grid vertices around an arbitrary particle position. For this, we extended a differentiable point cloud projector [Insafutdinov and Dosovitskiy, 2018] to arbitrary grid resolution, neighborhood size and custom kernel functions. Given all the neighboring particles $j \in \partial\Omega_{\mathbf{x}}$ around a grid node $\mathbf{x}$, a grid attribute $\lambda^{\boxplus}$ is computed by summing up weighted particle contributions as

$$\lambda^{\boxplus}(\mathbf{x}) = \frac{\sum_{j\in\partial\Omega_{\mathbf{x}}} \lambda_j^{\circ}\, W(||\mathbf{x}-\mathbf{x_j}^{\circ}||, h)}{\sum_{j\in\partial\Omega_{\mathbf{x}}} W(||\mathbf{x}-\mathbf{x_j}^{\circ}||, h)}, \tag{5.2}$$

where we chose W to be the cubic B-spline kernel, which is also often used in SPH simulations [Monaghan, 2005]:

$$W(r,h)_{\text{cubic}} = \begin{cases} \frac{2}{3} - r^2 + \frac{1}{2}r^3, & 0 \le r \le 1, \\ \frac{1}{6}(2-r)^3, & 1 \le r \le 2, \\ 0, & r > 2. \end{cases} \tag{5.3}$$

We now have all the necessary elements to convert the previous Eulerian style transfer (Equation (4.2)) into a Lagrangian framework. Given a set of Lagrangian attributes $\mathbf{\Lambda}^{\circ}$, the optimization objective for a single frame is

$$\hat{\mathbf{\Lambda}}^{\circ} = \underset{\mathbf{\Lambda}^{\circ}}{\arg\min} \sum_{\theta\in\Theta} \sum_{\lambda^{\circ}\in\Lambda^{\circ}} w_{\lambda^{\circ}}\, \mathcal{L}(\mathcal{R}_{\theta}(\mathcal{I}_{p2g}(\mathbf{x}^{\circ}, \lambda^{\circ}), \mathbf{p}), \tag{5.4}$$

where $w_{\lambda^{\circ}}$ are weights for the losses that include Lagrangian attributes. In case of particle position $\mathbf{x}^{\circ}$ given as the target quantity λ°, we use the SPH density $\mathcal{I}_{p2g}(\mathbf{x}^{\circ}) = \sum_{j\in\partial\Omega_{\mathbf{x}}} m_j W(||\mathbf{x}-\mathbf{x}_j^{\circ}||, h)$, where m_j represents the mass of the j-th particle [Bender, 2016]. Note that our losses are evaluated similarly as in the original Eulerian method, since the gradients computed in image-space also modify grid values ($\lambda^{\boxplus}$). However, these gradients are automatically propagated back to the particles by auto-differentiating the particle-to-grid $\mathcal{I}_{p2g}$ function. Thus, our method only reformulates the domain of the optimization, sharing the same stylization possibilities (semantic and content transfers) as in the original TNST.

Since the Lagrangian optimization is completely oblivious to the underlying solver type, the chosen attributes for creating stylizations can be arbitrarily combined, enabling a wide range of artistic manipulations in different scene setups. We outline two strategies and demonstrate their impact on the stylization. The first one is particularly suitable for participating volumetric data, which are often simulated with grid-based solvers. It involves optimizing a scalar value carried by the Lagrangian stylization particles by Equation (5.4). For most of our smoke scenes, this scalar value is the density, though it can also be the color or emission. The regularization term

$$\mathcal{L}(\lambda^{\circ})_{\rho} = (\sum \Delta\lambda^{\circ})^2 - \sum \log ||\Delta\lambda^{\circ}||_1 \tag{5.5}$$

reinforces the conservation of the original amount of smoke. It minimizes the total net smoke change, preventing the stylization to undesirably fade out particles and keeping changes non-zero by minimizing cross-entropy loss at the same time. Figure 5.5 demonstrates the impact of different regularizer weights.

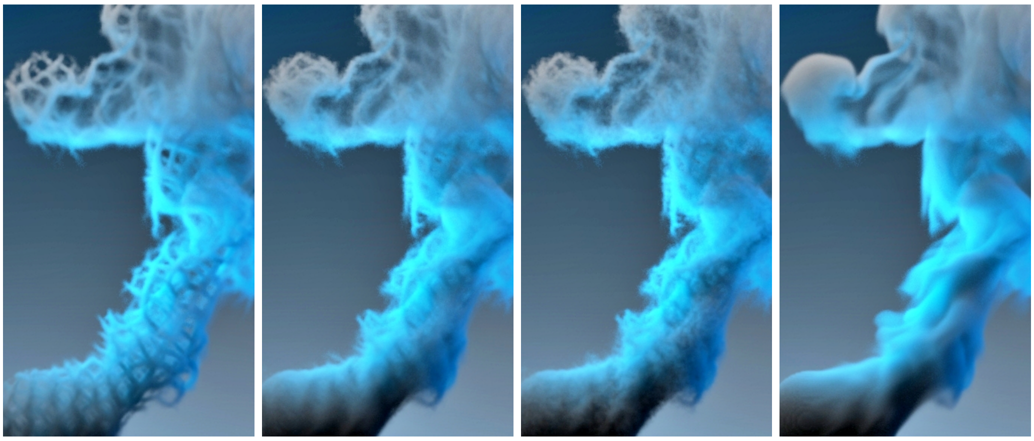


Figure 5.5: *Different weights for the density regularization show the trade-off between pronounced structures and conservation of mass. The images on the left show results with zero, low, and high weights, respectively, and the right image is the ground truth.*

The second strategy is suitable if the underlying fluid solver is particle-based or hybrid, which is often the case for liquids. For these simulations, we can define particle position displacements as the optimized Lagrangian attributes. However, generating stylizations by modifying particle displacements may cause cluttering or regions with insufficient particles. The regularization penalizes irregular distribution of particle positions and is defined as

$$\mathcal{L}(\mathbf{x}^{\circ})_{\Delta x} = ||\mathcal{I}_{p2g}(\mathbf{x}^{\circ}) - \rho_0^{\boxplus}||_2^2, \tag{5.6}$$

where $\rho_0^{\boxplus}$ corresponds to the rest density for cells that contain particles, and is zero otherwise. Note that Equation (5.6) does not account for the particle deficiency near fluid surfaces. This could be addressed by adding virtual particles [Schechter and Bridson, 2008] or applying (variants of) the Shepard correction to the kernel function [Reinhardt et al., 2019]. We show the impact of this regularizer on the particle sampling in Figure 5.6, highlighting the trade-off between uniform distribution and stylization strength.

We notice that both regularizations in Equation (5.5) and Equation (5.6) are different incarnations of the mass conservation property commonly used in

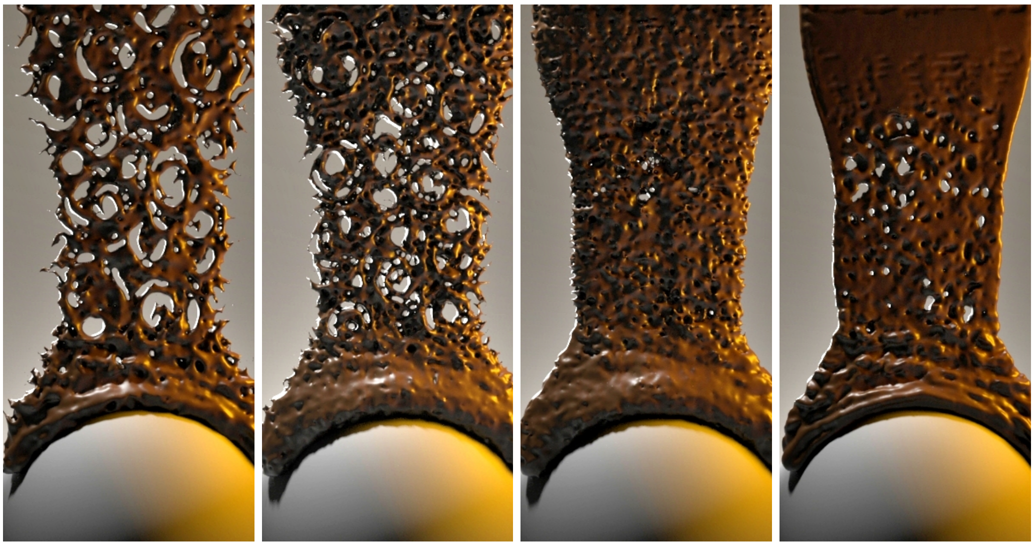


Figure 5.6: *Different weights for the position regularization show the trade-off between pronounced structures and uniform sampling. The images on the left show results with zero, low, and high weights, respectively, and the right image is the ground truth.*

fluid simulations. In TNST, mass conservation is enforced by decomposing the stylization velocities into their irrotational and incompressible parts, which can be optimized independently. Both techniques enable a high degree of artistic control over the content manipulation.

5.3 An Efficient Particle-Based Smoke Re-Simulation

If the input is a grid-based simulation, we have to sample and re-simulate particles. We can use a sparse representation with only one particle per voxel, in constrast to hybrid liquid simulations that usually sample 8 particles per voxel to properly capture momentum conservation [Zhu and Bridson, 2005]. Combining a low number of particles with a position integration algorithm that accumulates errors over time will yield irregularly distributed particles [Ando and Tsuruno, 2011]. This manifests in a rendered image as smoke with overly dense or void regions. We therfore solve the following optimization problem

$$\hat{\mathbf{x}}^\circ, \hat{\rho}^\circ = \underset{\mathbf{x}^\circ, \rho^\circ}{\arg\min} \sum_t ||\mathcal{I}_{p2g}(\mathbf{x}_t^\circ, \rho_t^\circ) - \rho_t^{\boxplus}||_2^2. \tag{5.7}$$

The optimization problem presented above is not only severely under-constrained but also has a time-varying objective term, and optimizing for

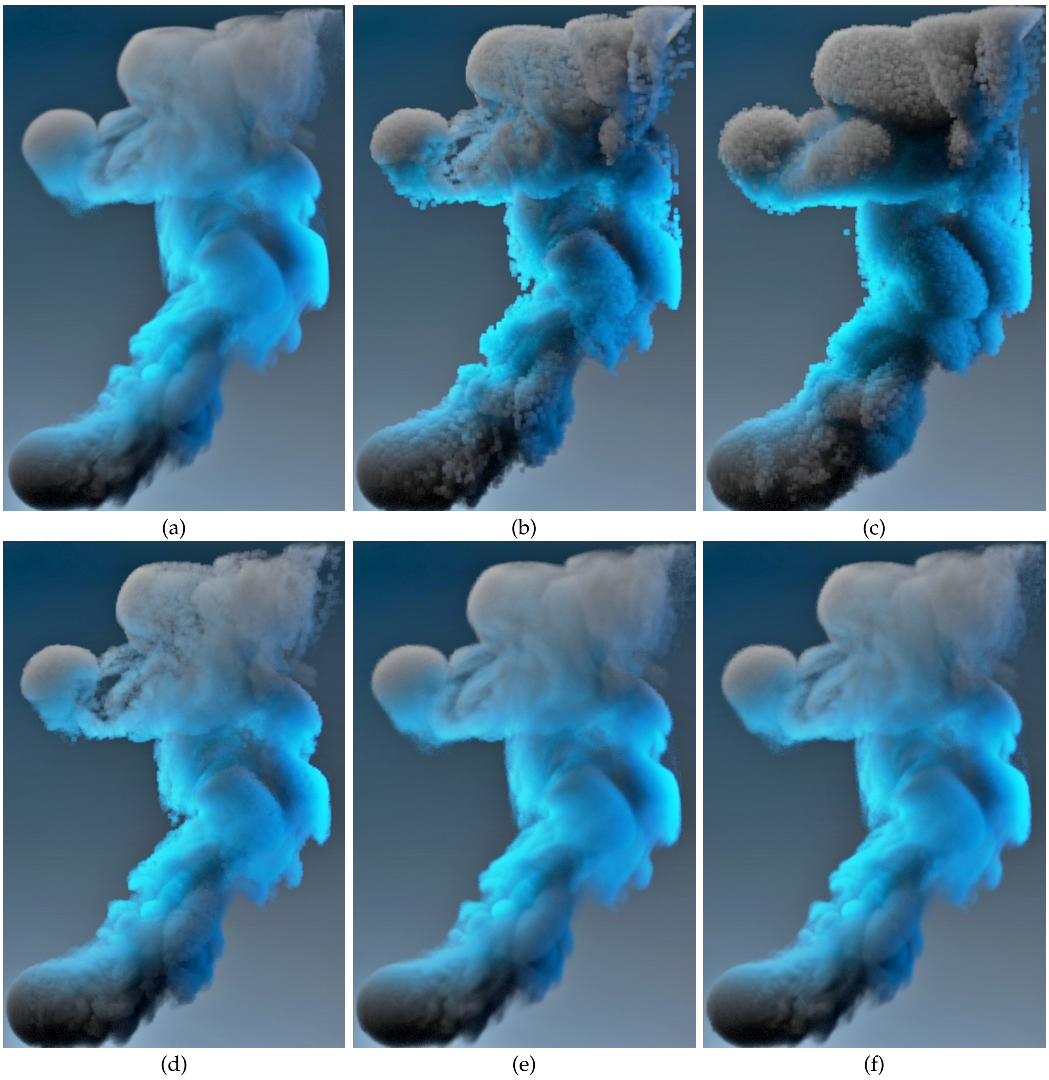
Equation (5.7) is challenging if tackled jointly for both particle positions $\mathbf{x}^\circ$ and densities ρ°. Thus, we use a heuristic approach for solving this optimization, subdividing it into two steps, position optimization and multi-scale density update (Section 5.3.1). Firstly, we minimize the irregular distribution of particle positions by employing a position-based update, optimizing particle distributions using Equation (5.6) as objective. The distribution of the particles is optimized per frame and serves as an input for optimizing subsequent frames, enabling temporally coherent position updates. Equation (5.6) can be automatically computed by our fully differentiable pipeline.

5.3.1 Multi-Scale Density Representation

In addition to the position update, we also compute smoke densities individually carried by the particles to further eliminate small gaps that may appear due to the sparse discretization, further enhancing the solution of Equation (5.7). Owing to the low number of sampled particles and the mismatches between grid and particle transfers, carrying a constant density will either produce grainy (Figure 5.7, (b)) or diffuse (Figure 5.7, (c)) volumetric representations, depending on if particle re-distribution (Equation (5.6)) is applied or not. A simple approach is to interpolate density values directly from the grid over time. Larger kernel sizes could be used to remedy sparse sampling, but would excessively smooth structures and degrade quality.

We take inspiration from Laplacian pyramids, where distinct grid resolution levels are treated separately. In our case, we compute residuals of different support kernel sizes of the particle-to-grid transfer. This efficiently captures both low- and high-frequency information, covering potentially empty smoke regions while also providing sharp reconstruction results. The residual computation of kernels of varying support sizes is synergistically coupled with matching grid resolutions, which creates an efficient multi-scale representation of the smoke.

The multi-scale reconstruction works as follows: we first sample grid densities to the particles. This represents the smoke low-frequency information, which we interpolate to the particle variables ρ_0°. The variables above the first level (e.g., $\rho_1^\circ, \rho_2^\circ$) will carry residual information computed between subsequent levels. The Lagrangian representations vary between each level because they perform grid-to-particle transfers with progressively reduced kernel support sizes. To compare residuals between Lagrangian representations, we make use of particle-to-grid transfers, which act as a low-pass filter, similarly to blurring operations of Laplacian pyramids. This process is performed until the original grid resolution is matched. Our multi-scale density



(a) (b) (c)

(d) (e) (f)

Figure 5.7: *Comparison of different re-simulation strategies. (a): ground truth density, (b): constant density carried by particles, (c): (b) with redistribution by Equation (5.6), (d): single-scale sampled density, (e): (d) with redistribution, (f): multi-scale ($n_s = 3$) sampled density with redistribution (final method).*

representation is summarized in Algorithm (3). Figure 5.7 illustrates the impact of using a single scale without (d) and with (e) particle re-distribition (Equation (5.6)). The multi-scale result with re-distribution (f), which corresponds to our final method, has a higher PSNR (31.89) than its single-scale counterpart (31.39) and is very close to the ground truth (a).

Algorithm 3: *Multi-Scale Density Reconstruction*

Data: Particle positions $\mathbf{x}^{\circ}$ optimized by Equation (5.6)
Original grid-based smoke simulation $\rho^{\boxplus}$
Grid node positions $\mathbf{x}^{\boxplus}$
Coarsest support kernel radius r
Number of pyramid subdivisions n_s
Result: Multi-scale residual density $\rho^{\circ} = \{\rho_i^{\circ}\}_{i=1..n_s}$ stored on particles

```
$\rho_0^{\boxplus} \leftarrow 0$    // Initialize particle density reconstruction
$r_1 \leftarrow r$    // Initialize kernel radius
for $i \leftarrow 1$ to $n_s$ do
    $\rho_r^{\boxplus} \leftarrow \rho^{\boxplus} - \rho_{i-1}^{\boxplus}$    // Compute residual in grid representation
    $\rho_i^{\circ} \leftarrow \mathcal{I}_{g2p}(\mathbf{x}^{\circ}, \rho_r^{\boxplus})$    // Particle residual sampling
    $\rho_i^{\boxplus} \leftarrow \rho_{i-1}^{\boxplus} + \mathcal{I}_{p2g}(\mathbf{x}^{\circ}, \rho_i^{\circ}, r_i, \mathbf{x}^{\boxplus})$    // Update density reconstruction
    $r_{i+1} \leftarrow \frac{r_i}{2}$    // Divide the kernel radius in half
end for
```

5.4 Temporal Coherency

The major advantage of our Lagrangian discretization is the inexpensive enforcing of temporal coherency. Since quantities are carried individually per particle, it is intrinsically simple to track how attributes change over time. Neural style gradients are computed on the grid and need to be updated once the neighborhood of a particle changes. To ensure smooth transitions, we apply a Gaussian filter over the density changes of a particle, as shown in Figure 5.8. Besides being sensitive to density neighborhood changes, stylization gradients are also influenced by the density carried by the particle itself (Section 5.3.1).

To further improve efficiency, and in contrast to TNST, we can keyframe stylizations, i.e., apply stylization to keyframes and interpolate particle attributes in-between. In practice, we reduced the stylization frames by a factor of 2 at max, but more drastic approximations could be used. Sparse keyframes still show temporally smooth transitions, but quality is degraded. Nevertheless, sparse keyframing would still be useful for generating quick previews of the simulation. The impact of sparse keyframing (every 10 frames) is shown in Figure 5.9.

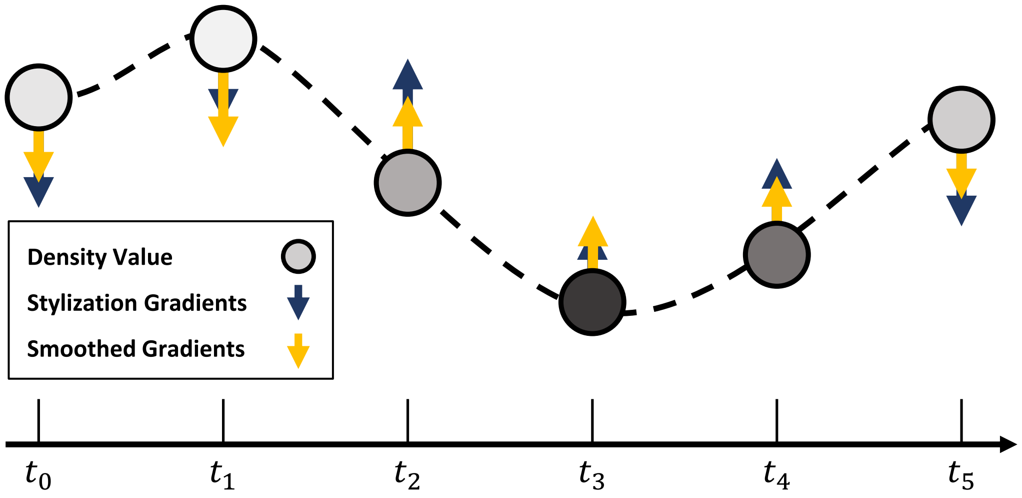


Figure 5.8: *Particle density (circles) variation for a single particle over time. Temporal coherency is enforced by smoothing density gradients used for stylization from adjacent frames.*

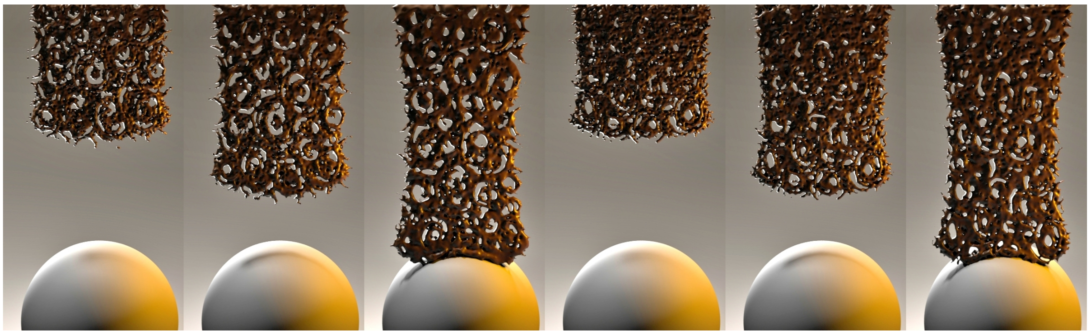


Figure 5.9: *Stylization of every frame (left three images) versus keyframed stylization every 10 frames (images on the right). Sparse keyframing is visually similar and can be useful for quick previews.*

5.5 Results

We implemented the method with the *tensorflow* [Abadi et al., 2016] framework and computed results on a TITAN Xp GPU (12GB). We used *mantaflow* [Thuerey and Pfaff, 2018] for smoke scene generation, a 3D smoke dataset from Kim et al. [2019a] for comparisons with TNST, a 2D smoke dataset from Jamriška et al. [2015] for color stylization, *SPlisHSPlasH* [Bender, 2016; Koschier et al., 2019] for liquid simulations and *Houdini* for rendering.

Performance Using particles for stylization eliminates the need for recursively aligning stylization velocities from subsequent frames, which notably improves the computational performance. In combination with our sparse particle respesentation for smoke (1 particle per cell), simulations of size 200 × 300 × 200 can now be stylized within an hour instead of a day (TNST). The computation time per frame is 0.66 minutes for the Smoke Jet scene shown in Figure 5.11, which is a speed-up of a factor of 20.41 compared to TNST. This improvement allows artists to more easily test different reference structures (input images) and hence renders neural flow stylization better applicable in production environments. Table (5.1) gives an overview of the timings and parameters for the individual test scenes. Keyframing (every other frame) was applied to the Smoke Jet (Figure 5.11) and Double Jets (Figure 5.12) examples.

Table 5.1: *Performance table.*

Scene	Resolution	# Particles	Time (m/f)
Moving Sphere (Fig. 5.10)	192 × 192 × 192	237K	0.8
Smoke Jet (Fig. 5.11)	200 × 300 × 200	1.2M	0.66
Double Jets (Fig. 5.12)	200 × 200 × 200	2M/2M	0.45
Chocolate (Fig. 5.13)	200 × 200 × 200	80K	0.05
Colored Smoke (Fig. 5.2)	800 × 800	136K	1.21
Dam Break (Fig. 5.14)	512 × 1024	23K	0.58
Double Dam (Fig. 5.15)	512 × 1024	31K/8K	0.65

Time-Coherency To illustrate the benefit of the Lagrangian formulation, we use a simple test scene where we initialize a smoke sphere with a uniform density. We then move the smoke artificially to the right, and apply the neural stylization to every frame of the sequence. We compare the results of LNST and TNST for different time instances in Figure 5.10. The top row shows the results of TNST. It can be seen that TNST is not able to preserve constant stylized textures in regions where the density function does not change. This is due to the recursive alignment of stylization gradients, which accumulate errors especially for bigger window sizes. The second row shows the corresponding results with LNST, demonstrating consistent stylization over time since gradients are constant. Also when applied a shearing deformation to the sphere, as shown in the third row, strucutures remain coherent. If an artist prefers to have changing structures in such situations, noise can be added to the densities carried by the particles, which in turn will induce stylization gradients as shown in the last row.

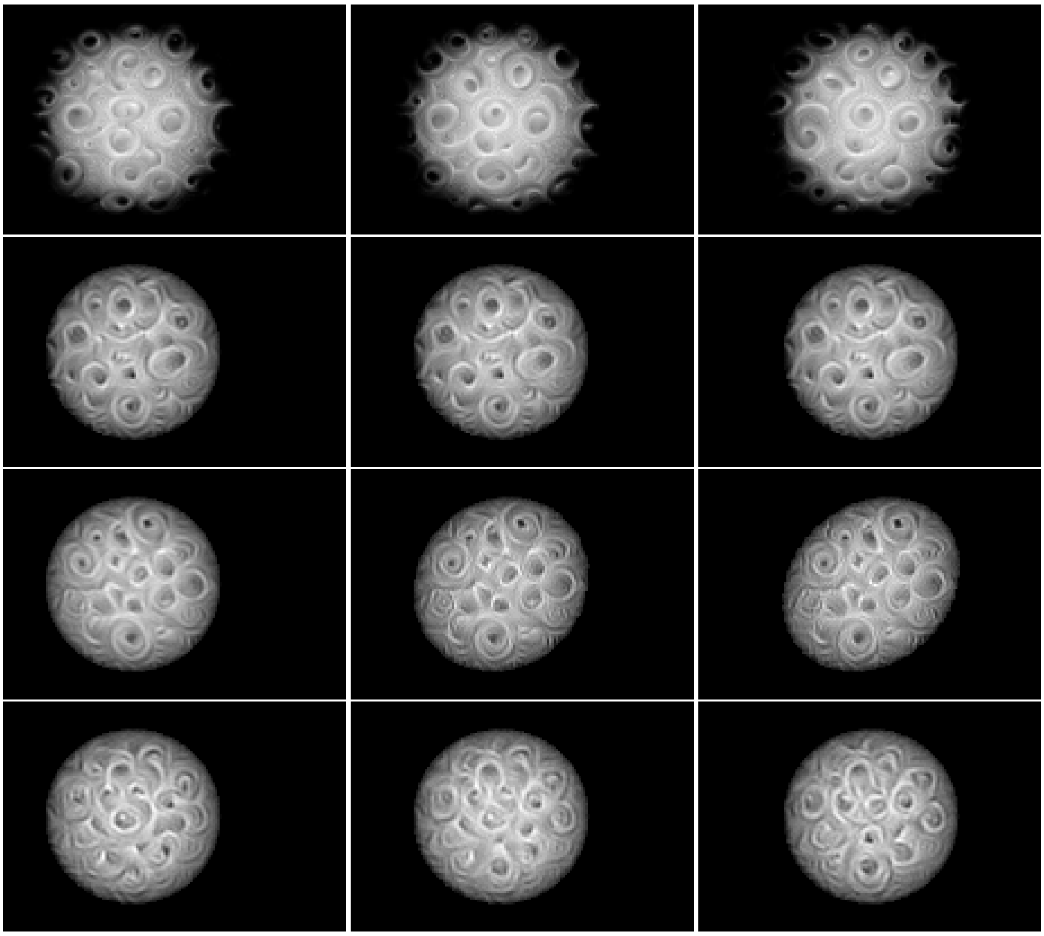


Figure 5.10: *Selected frames of a stylized moving smoke sphere. From top to bottom: TNST with structures changing over time, LNST with temporally coherent structures, LNST result with applied shearing, and LNST result with noise-added density inducing style variation over time.*

Smoke Stylization Figure 5.11 shows a direct comparison of LNST and TNST applied to the smoke jet dataset of Kim et al. [2019a]. While the resulting structures inherently depend on the underlying representation, they naturally differ and cannot be directly compared with each other. It can be observed, however, that the Lagrangian stylization may lead to more pronounced structures, well visible in the semantic transfer *net* and the style transfer *blue strokes*, and that boundaries are smoother, noticeable in the *Seated Nude* example.

Multi-Fluid Stylization Stylization of multiple fluids is naturally enabled by stylizing different sets of particles with different input images. Figure 5.12 shows a simulation of two smoke jets colliding, where the left one is

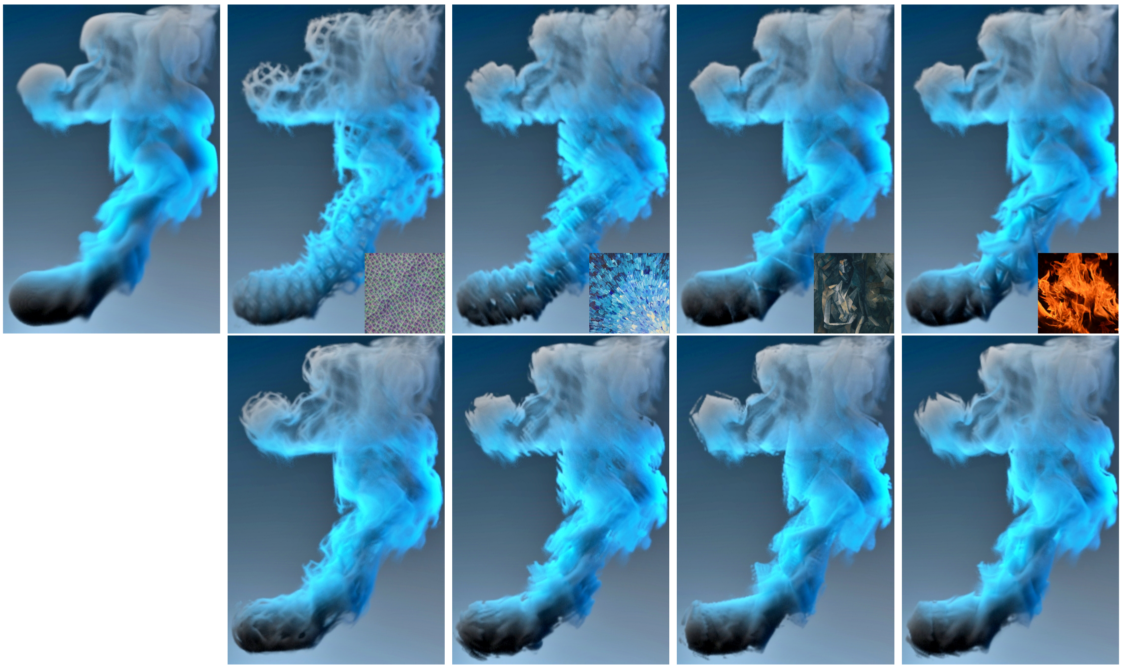


Figure 5.11: *Semantic transfer applied to the smokejet simulation of [Kim et al., 2019a] (leftmost column). Stylized results are shown for our LNST (top) and TNST (bottom) for semantic feature transfer* net *(second column) and input images* blue strokes, seated nude, *and* fire *(last three columns) [gfv, 2015; sti, 2018].*

stylized with the semantic feature *net* and the right one with the style transfer of the input image *spirals*. Transferred structures are retained per fluid type even if the flow undergoes complex mixing effects.

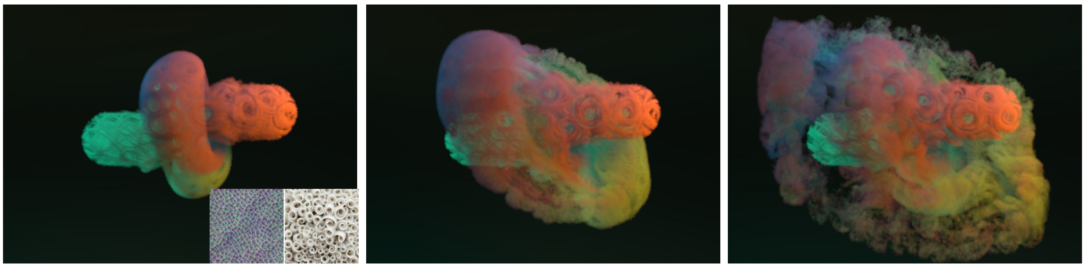


Figure 5.12: *Two colliding smoke jets, which are stylized individually with the semantic feature* net *and input image* spirals. *The Lagrangian representation enables coherent stylization of multiple fluids even if the flow undergoes complex mixing.*

Stylization of Liquids We use a simple differentiable renderer for stylization of liquids. Unlike smoke renderer, which integrates media radiance scattered in the medium, we compute the amount of diffused light, i.e., absorbed light except transmitted by its liquid volume [Ihmsen et al., 2012], which is given by

$$\begin{aligned} \tau(\mathbf{x}, \mathbf{r}) &= e^{-\gamma \int_{\mathbf{0}}^{\mathbf{x}} d(\mathbf{r})\, dr} \\ I_{ij} &= 1 - \tau(\mathbf{r}_{max}, \mathbf{r}). \end{aligned} \tag{5.8}$$

Figure 5.13 shows the results of a stylized SPH simulation computed with *SPlisHSPlasH* [Bender, 2016]. We applied the patterns *spiral* and *diagonal* to a thin sheet simulation.

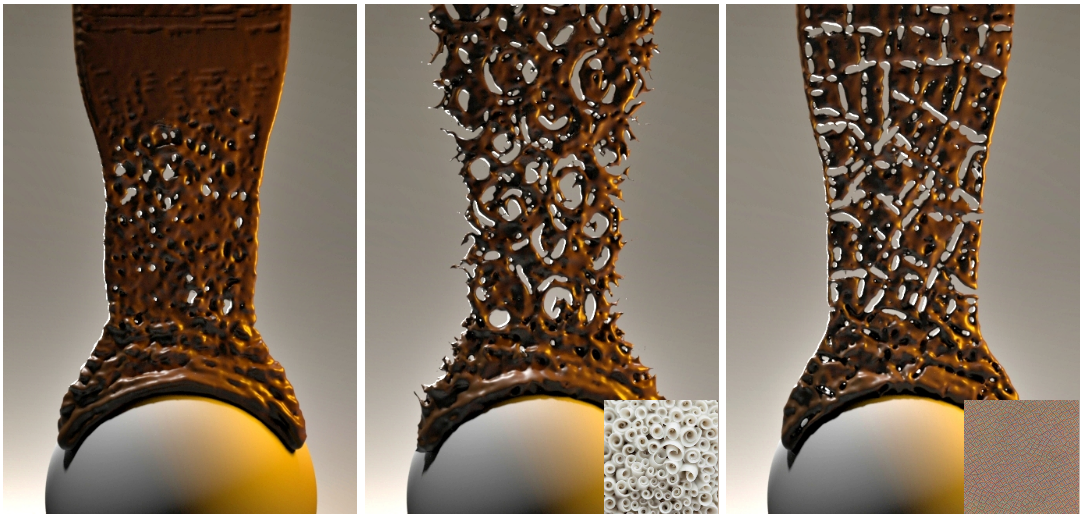


Figure 5.13: *Thin sheet SPH simulation computed with* SPlisHSPlasH *[Bender, 2016] stylized with the patterns* spiral *and* diagonal.

Color Transfer We transfer color information from input images to flow fields by storing a color value per particle and optimizing it by Equation (5.4). This can be applied to any grid-based or particle-based smoke or liquid simulation. In Figure 5.14 we applied the color stylization to a 2D dam break simulation using different example images, and in Figure 5.15 to two liquids with distinct types (and hence color). The accompanying videos show that local color structures change very smoothly over time, which is attributed to the improved time-coherency of the Lagrangian stylization. This is especially well visible in Figure 5.2, where two subsequent frames are shown for TNST and LNST. In this example, we have transferred the style *Red Canna* to a smoke scene. The close-up views reveal discontinuities for TNST, while LNST shows smooth transitions for color structures.

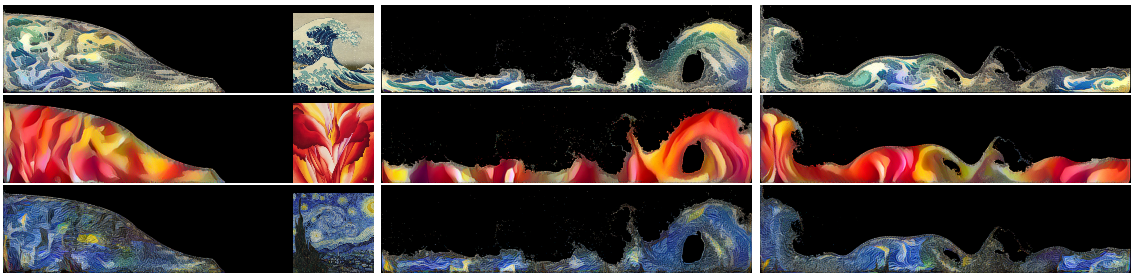


Figure 5.14: *Lagrangian color stylization applied to a 2D particle-based liquid simulation using the input images* Kanagawa Wave, Red Canna *and* Starry Night.

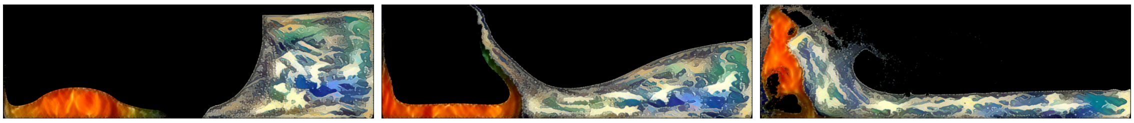


Figure 5.15: *Lagrangian color stylization applied to a* mixed *2D particle-based liquid simulation using the input images* Kanagawa Wave *and* fire.

CHAPTER 6

Deep Reconstruction of 3D Smoke Densities from Artist Sketches

This chapter is based on the following submitted publication:

X. Huang, G. Cordonnier, B. Kim, V. C. Azevedo, M. Gross, B. Solenthaler, **Deep Reconstruction of 3D Smoke Densities from Artist Sketches**, *submitted to SIGGRAPH Asia 2020.*

The author primarily contributed to *Section: 2D Sketch to 3D Density Prediction* and overall experiment design, during his joint PhD at DisneyResearch|Studios.

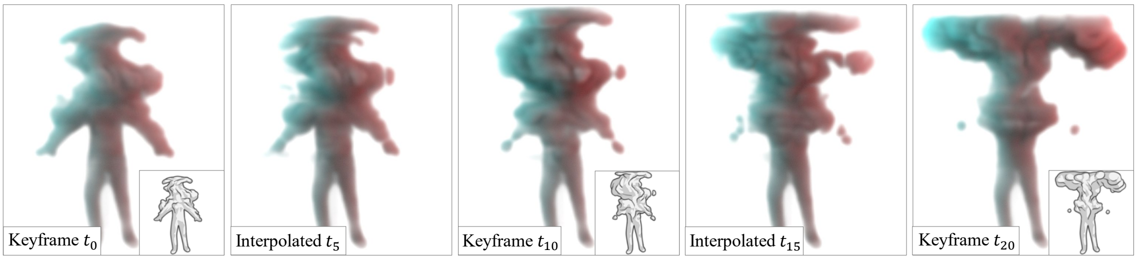


Figure 6.1: *Selected frames of the* dissolving character *example. Our network takes as input an artist sketch at keyframes t_0, t_{10}, and t_{20}, and computes the corresponding 3D smoke fields. We compute in-between frames t_5 and t_{15} with an interpolation method based on Wasserstein barycenters.*

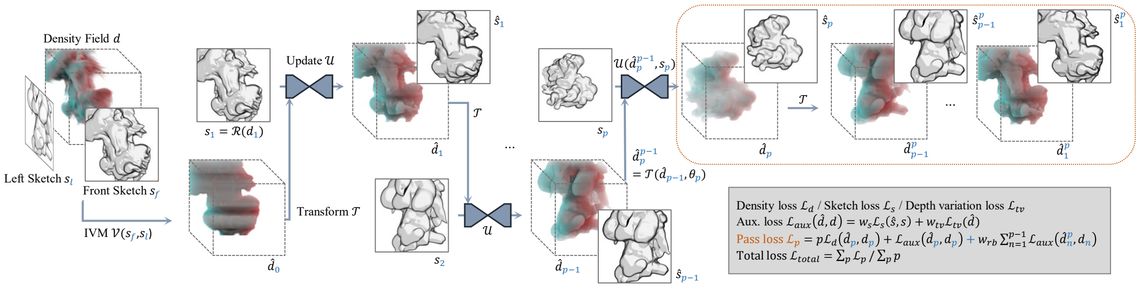

Density Field d
Left Sketch s_l
Front Sketch s_f
IVM $\mathcal{V}(s_f,s_l)$
$\hat{d}_0$
$s_1 = \mathcal{R}(d_1)$
Update $\mathcal{U}$
Transform $\mathcal{T}$
$\hat{s}_1$
$\hat{d}_1$
$\mathcal{T}$
s_2
$\mathcal{U}$
...
s_p
$\mathcal{U}(\hat{d}_p^{p-1}, s_p)$
$\hat{d}_p^{p-1} = \mathcal{T}(\hat{d}_{p-1}, \theta_p)$
$\hat{d}_{p-1}$
$\hat{s}_{p-1}$
$\hat{s}_p$
$\hat{d}_p$
$\mathcal{T}$
$\hat{s}_{p-1}^p$
$\hat{d}_{p-1}^p$
...
$\hat{s}_1^p$
$\hat{d}_1^p$
Density loss $\mathcal{L}_d$ / Sketch loss $\mathcal{L}_s$ / Depth variation loss $\mathcal{L}_{tv}$
Aux. loss $\mathcal{L}_{aux}(\hat{d}, d) = w_s\mathcal{L}_s(\hat{s}, s) + w_{tv}\mathcal{L}_{tv}(\hat{d})$
Pass loss $\mathcal{L}_p = p\mathcal{L}_d(\hat{d}_p, d_p) + \mathcal{L}_{aux}(\hat{d}_p, d_p) + w_{rb}\sum_{n=1}^{p-1}\mathcal{L}_{aux}(\hat{d}_n^p, d_n)$
Total loss $\mathcal{L}_{total} = \sum_p \mathcal{L}_p / \sum_p p$

Figure 6.2: *Overview of the multi-pass/view training for sketch to density reconstruction. Our pipeline takes a single density field and generates corresponding sketches for the end-to-end training. At each pass of the updater network, a pass loss $\mathcal{L}_p$ is computed on intermediate results (orange dashed box) to match all target sketches.*

6.1 Overview

In the previous chapters, we have presented the first generative model that constructs a wide variety of fluid behaviors from a set of reduced parameters, and we have also presented novel neural style transfer methods for fluids, which takes flow appearance and motion control to a new level. While those methods provide artists an efficient tool for artistic fluid control, their creative processes often start with sketches illustrating an object and its motion over time. Reproducing these sketched keyframes as 3D density clouds that capture realistic flow details is highly non-trivial and remains a manual and time-consuming process.

Therefore, in this Chapter 6, we aim to compute a 3D density field directly from a set of 2D artist sketches. The proposed method bridges the gap between early-stage prototyping of smoke keyframes and visual realization by inferring 3D densities from sketch inputs.

Sketch-based reconstruction has to handle a set of unique challenges, as the problem is ill-posed due to sparsity of sketch data and the ambiguity when inferring another dimension [Delanoy et al., 2018; Wang et al., 2019; Li et al., 2018a]. We propose a convolutional neural network (CNN) approach that synergistically combines a U-net architecture with a differentiable sketch renderer and a set of loss functions specifically designed for our application.

Taking two sketch inputs representing the front and left views, we compute an initial volume estimate and refine the density by turns with a convolutional neural network. During training, our pipeline takes a single density field and generates corresponding sketches for the end-to-end training. At each pass, we pick a random view for training and calculate view-dependent losses. The updater network iteratively refines the reconstructed density. A pass loss is computed on intermediate results to match all target sketches. These passes are alternatively applied until convergence to ensure multi-view consistency.

At the core of our method is a differentiable sketch generator for smoke volumes, which allows us to compute a sketched representation of the reconstructed density and compare it to the input sketch by a sketch loss.

To compute an animated sequence, we interpolate between a pair of reconstructed density keyframes using Wasserstein barycenters. This approach is based on the theory of optimal transport and allows us to reconstruct intermediate densities at any frame without needing to compute velocities or previous frames (Figure 6.1).

Our training dataset captures a wide range of different smoke simulations and sketch style augmentations to ensure robustness and generic application of the approach. We present results on synthetic datasets, such as physically-based smoke scenes and other (non-smoke) animations. We further apply the method to sketches of smoke drawn by different artists, which were instructed to sketch keyframes of the evolution of a smoke or an imaginary smoke-like object. The motion could disobey physical laws, where volume is not conserved, and inconsistencies may appear with respect to shading, shape and motion in subsequent keyframes. We demonstrate that our model can handle even such extreme cases, although they were not part of the training set. The ablation study evaluates our network components' choices, and final animation sequences are shown to highlight the use cases of our novel deep sketch to density pipeline.

6.2 2D Sketch to 3D Density Prediction

Similarly to recent sketch-based 3D reconstruction methods for static geometries [Delanoy et al., 2018], we aim to construct a network that can infer 3D smoke densities from a few 2D artist sketches. The sketches are the primary control tool for the artist, and they should give enough information on the outline of the smoke and smaller-scale details. Therefore, we impose a sketch style showing the outline of the smoke as well as the internal contour, and we add one shade of toon-shading from an infinite light to have a better representation of the volume as shown, for example, in Figure 6.1.

At the core of our method is a differentiable sketch generator (Section 6.3) for smoke volumes, which allows us to compute a sketched representation of

Table 6.1: *Summary of the notation.*

Notation	Description
d	Target density field
$d_p = \mathcal{T}(d, \sum_{i=1}^{p} \theta_i)$	Rotated density field
$s_p = \mathcal{R}(d_p)$	Sketch of the density field
$\hat{d}_0 = \mathcal{V}(s_f, s_l)$	Initial density (initial volume modeling)
$\hat{d}_p = \mathcal{U}(\hat{d}_p^{p-1}, s_p)$	Refined density field after pass p
$\hat{d}_p^{p-1} = \mathcal{T}(\hat{d}_{p-1}, \theta_p)$	Rotated density field after pass $p-1$
$\hat{d}_{p-1}^{p} = \mathcal{T}(\hat{d}_p, -\theta_p)$	Roll-backed density field after pass p
$\mathcal{L}_d, \mathcal{L}_s, \mathcal{L}_{tv}$	Density, sketch and variation losses (Eqs. 6.1,6.2,6.3)
$\mathcal{L}_{aux}, \mathcal{L}_p, \mathcal{L}_{total}$	Auxiliary, pass and total losses (Eqs. 6.4,6.5,6.6)

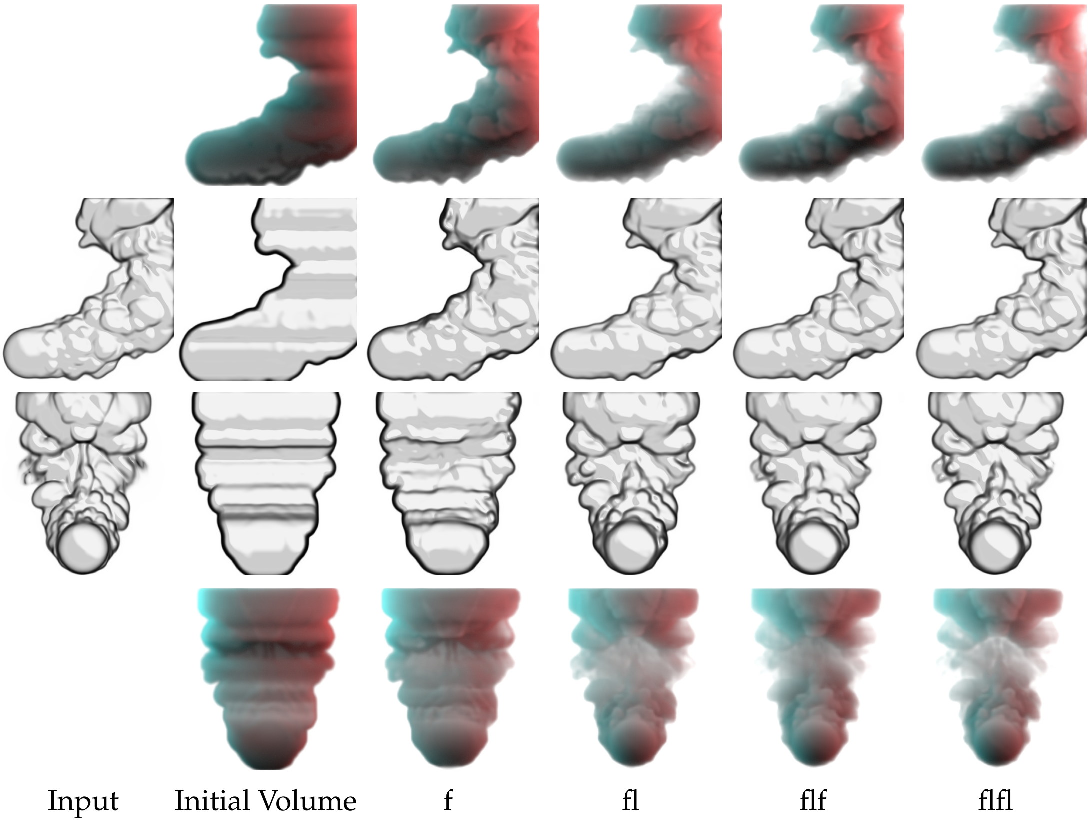


Figure 6.3: *From two input sketches depicting the front and left views, we compute the initial volume and then alternately refine front (f) and left (l) view density reconstructions.*

the density reconstruction and to compare this to the input sketch by using a sketch loss. An additional advantage of the synthetic sketcher is that it allows us to build a large dataset of smoke simulations coupled with sketches inexpensively (Section 6.5).

Our approach requires two input sketches s_f and s_l indicating the front and left views of the smoke. These are used to compute an initial density $\hat{d}_0$ by simple extrusion and intersection from multi-view sketches. We refer to this as Initial Volume Modeling (IVM) (Section 6.2.1) in the following. The density $\hat{d}_0$ is then progressively refined with an updater network, where arbitrary views can be added (Section 6.2.2). A pass loss is computed on intermediate results to match all target sketches (Section 6.2.3). These passes are iteratively applied until convergence to ensure multi-view consistency. The final reconstructed density field is referred as $\hat{d}$. An illustration of subsequent refinement steps on front and side views is shown in Figure 6.3 (we used front-left (fl) refinement in all our examples). Our notation is summarized in Table (6.1).

6.2.1 Initial Volume Modeling

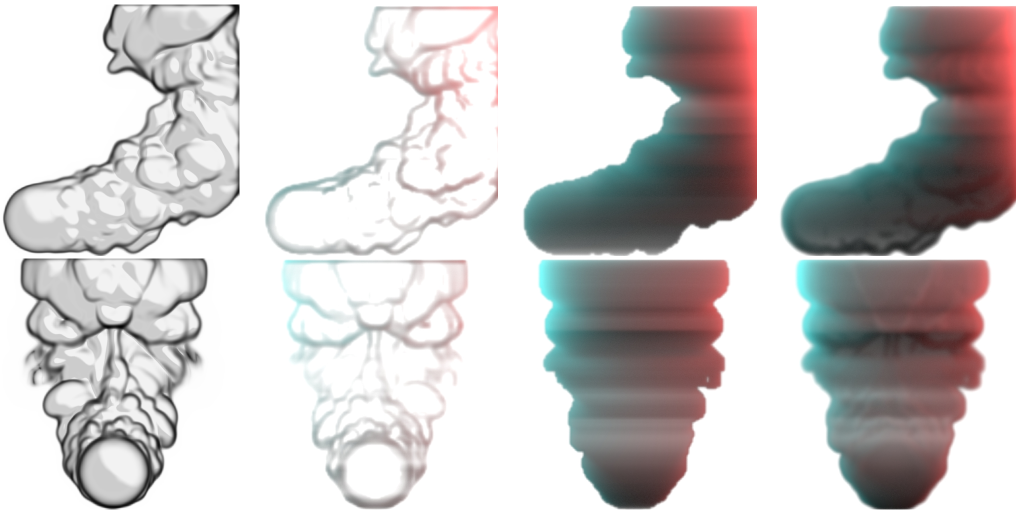


Figure 6.4: *Steps of the initial volume modeling shown for the front (top) and left (bottom) views. From left to right: input sketch, outline, binary, blended and smoothed volume ($\hat{d}_0$, and input to the updater network).*

The initial volume modeling (IVM) computes a guess for the density field $\hat{d}_0$ from the input sketches. We used the front and left view sketches s_f and s_l, but an additional view from the top could be added if needed. For each sketch, we first extract the contours and foreground with thresholding and filling operations. We then extrude the sketch of that viewpoint to an outline 3D volume and a binary 3D volume. We calculate the intersections of these volumes from the different views, blend the outline and binary volumes and apply Gaussian blur to smooth the boundaries. The output represents the initial guess for the subsequent refinement steps, i.e., the input to the updater CNN $\mathcal{U}$. The individual processing steps are visualized in Figure 6.4. All steps are differentiable and thus included in the end-to-end training pipeline.

IVM has inherent advantages compared to computing the initial volume with a single-view CNN [Delanoy et al., 2018], since the computation is deterministic, the resulting topology is closed, no training is necessary, and it is independent of the quality of the network. These properties favor the convergence of the subsequent refinement steps.

6.2.2 CNN-Based Iterative Refinement

We adopt an updater CNN that iteratively fuses information from multiple views by taking an artist sketch and a previously reconstructed density field as input. In the first iteration, the predicted density field corresponds to the initial density $\hat{d}_0$ computed with IVM. During the optimization, the refined density $\hat{d}_p$ refers to the output after p passes. These passes are alternatively applied until convergence to ensure multi-view consistency. An overview of the training pipeline is shown in Figure 6.2. The concept of using an updater network is similar to Delanoy et al. [2018], but there are significant differences in the network components, loss functions and training process that impact reconstruction quality and robustness of predictions. We train the network to predict the residual [He et al., 2016] for correction instead of predicting the density field directly. Due to the ambiguity of line drawings, [Delanoy et al., 2018] focuses on a few informative viewpoints for training. Although we also use canonical viewpoints, our differentiable sketcher allows not only less ambiguity on depth with its shading but also an easy extension to arbitrary viewpoints on the fly, without any loss of generality.

We also improve the refinement strategy by including recursive passes already in the training step. In each pass, we pick a random view for training, as multi-pass/view training results in a higher robustness for the reconstruction when the density field is refined from multiple viewpoints. Specifically, we first apply random rotations $\mathcal{T}(d, \theta)$, where θ is the angle, to choose 1 of the 24 canonical views. Similarly, we choose random views for the subsequent rotations.

6.2.3 Loss Functions

For each view, we calculate the view-dependent losses: density loss, sketch loss, and depth variation loss, as well as rollbacks in each pass for preserving reconstructions of previous input views.

Density Loss We define our density loss $\mathcal{L}_d$ in terms of $L1$ loss on the reconstructed field, similar to [Kim et al., 2019b]:

$$\mathcal{L}_d(\hat{d}, d) = ||\hat{d} - d||_1, \tag{6.1}$$

where d refers to the ground truth density field and $\hat{d}$ is the reconstructed field from our refinement network.

Sketch Loss A main difference to previous work is the integration of a differentiable sketcher $\mathcal{R}$ into the network, which allows us to generate a target sketch $s_p = \mathcal{R}(d_p)$ and a sketch of the refined density $\hat{s}_p = \mathcal{R}(\hat{d}_p)$ at each refinement step p during training. We use this to minimize the distance to the input sketch with a sketch loss $\mathcal{L}_s$ given as

$$\mathcal{L}_s(\hat{s}, s) = ||\hat{s} - s||_1, \tag{6.2}$$

where s is the input sketch and $\hat{s}$ is the rendered sketch from the reconstructed density $\hat{d}$.

Depth Variation Loss As the sketch loss only focuses on the rendering of the 3D volume, there is no constraint in depth direction during training. Although the sketch shading encodes some depth information, artifacts in depth are still noticeable even with the density loss (Figure 6.14). We therefore use the total variation loss in depth direction as a regularizer to enforce smoothness:

$$\mathcal{L}_{tv}(\hat{d}) = ||\nabla_z \hat{d}||_1. \tag{6.3}$$

Full Objective We define the auxiliary loss $\mathcal{L}_{aux}$ combining sketch loss and total variation loss in depth as the following:

$$\mathcal{L}_{aux}(\hat{d}, d) = w_s \mathcal{L}_s(\hat{s}, s) + w_{tv} \mathcal{L}_{tv}(\hat{d}), \tag{6.4}$$

where w_s and w_{tv} are weights for the sketch loss and total variation loss set to 0.03 and 0.1, respectively. We define the pass loss $\mathcal{L}_p$ as

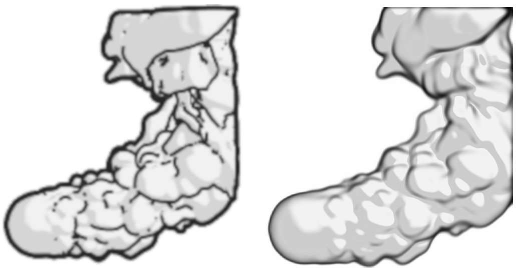
$$\mathcal{L}_p = p\mathcal{L}_d(\hat{d}_p, d_p) + \mathcal{L}_{aux}(\hat{d}_p, d_p) + w_{rb} \sum_{n=1}^{p-1} \mathcal{L}_{aux}(\hat{d}_n^p, d_n), \tag{6.5}$$

where w_{rb} is the rollback weight, and $\hat{d}_n^p$ denotes the density $\hat{d}_n$ rotated to the viewpoint of pass n. Using a high w_{rb} is important for preserving the view-dependent reconstructions of the previous viewpoints. We therefore progressively increase the weight from 1 to 100 during the first epoch for optimal results. The total loss is then defined as:

$$\mathcal{L}_{total} = \frac{\sum_p \mathcal{L}_p}{\sum_p p}. \tag{6.6}$$

6.3 Differentiable Sketcher

The aim of our sketcher is to generate a stylized representation of a smoke keyframe while conveying the nature of turbulent flows. Although artists have individual drawing styles, we identified the common use of black contours and solid blocks of color for shading and self-shadowing [Gilland, 2012]. These effects provide important visual cues about the shape, depth and lightning. Previous techniques typically extract contours from meshes [DeCarlo et al., 2003] (inset image, left) and cannot be directly applied to amorphous shapes such as smoke densities. Although isosurfaces can be extracted from the density grid and converted into a mesh, the choice of the isovalue is not obvious and typically low-density details are lost in the process. Therefore, for cartoon rendering of smoke, particles have been used and traced through the underlying fluid grid, which are then rendered as textured billboards without [Selle et al., 2004] and with [McGuire and Fein, 2006] shading and shadow effects.



In contrast to using tracer particles, we work directly with volumetric data and adapt the mesh contour extraction to amorphous smoke shapes (inset image, right). Our method is inspired by the absorption in smoke rendering and estimates a normal map. We average normals for a range of isosurfaces at the visible 'boundary' of the smoke. At each position $\mathbf{p}$ in the volume, the normal of the isosurface going through $\mathbf{p}$ is the gradient of the density field: $\nabla d(\mathbf{p})$. We integrate the normals along the viewing rays, while weighting them such that the normals closer to the boundary have a higher importance.

For a ray r, the normal $\mathbf{n}_r$ is given as

$$\mathbf{n}_r = \int_0^{\inf} \nabla d(r(s))\, w(r(s))\, \mathrm{d}s, \tag{6.7}$$

where s is the distance to the camera and w the weighting factor.

A possible choice for the weights is to use the transmittance of the smoke. Computing the averaged normal is then equivalent to the rendering of the smoke, where the density gradient is set as the light emission of each of the voxels. We found, however, that this strategy results in too smooth variations in normals, leading to blurred sketches. Therefore, we choose a stricter filtering strategy: Let $\tau(s)$ be the accumulated density along the ray: $\tau(s) = \int_0^s d(t)\,\mathrm{d}t$. We choose our weight as

$$w(s) = \xi^2 \cdot \tau(s)\, e^{-\xi \cdot \tau(s)}, \tag{6.8}$$

where $\xi = 5$ is a scale coefficient. This ensures that a small band of smoke close to the boundary contributes to the final normal, while discarding the low density voxels that are first encountered by the ray and which produce noisy normals.

We make two assumptions to discretize Equation 6.7. First, we assume that the gradient of the density is constant between two voxels, and second, that τ varies linearly in a small neighborhood. The normal can then be defined as

$$\begin{aligned} \mathbf{n}_r &= \sum_{i=0}^{i=n-1} \frac{\nabla d(i) + \nabla d(i+1)}{2} \int_i^{i+1} w(\tau(s))\,\mathrm{d}s + \mathbf{n}_\infty \\ &= \sum_{i=0}^{i=n-1} \frac{\nabla d(i) + \nabla d(i+1)}{2} \frac{[-(\xi \cdot \tau + 1) \cdot e^{-\xi \cdot \tau}]_{\tau(i)}^{\tau(i+1)}}{\tau(i+1) - \tau(i)} + \mathbf{n}_\infty, \end{aligned} \tag{6.9}$$

where n is the number of cells along the ray (here it is assumed to have a cell size of 1, without loss of generality). n_∞ is the background normal, that we set opposed to the view direction:

$$\mathbf{n}_\infty = -\mathbf{v} \int_n^\infty w(\tau(s))\,\mathrm{d}s = -\mathbf{v}\,(\xi \cdot \tau(n) + 1) \cdot e^{-\xi \cdot \tau(n)}. \tag{6.10}$$

The integral appearing in Equation 6.10 is also used as a mask that indicates if the ray intersected the smoke.

We normalize $\mathbf{n}_r$, which is then used to extract the contours c as:

$$c = max(0, min(-\mathbf{n}_r \cdot \mathbf{v}, \delta))/\delta, \tag{6.11}$$

where δ is a threshold set to 0.8. We then add a two colors toon shade t to provide more information about the volumes, and we combine all to compute the sketch s as

$$s = (1 - c) \cdot c + c \cdot t. \tag{6.12}$$

The individual steps of the sketcher $\mathcal{R}$ are visualized in Figure 6.5.

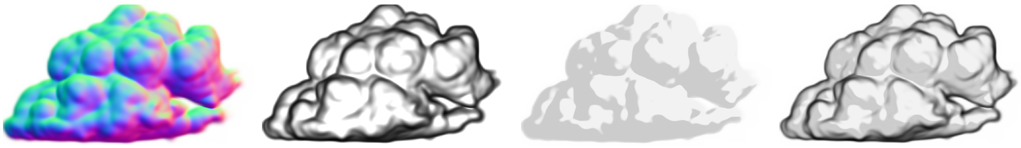


Figure 6.5: *The differentiable sketcher computes normals, contours and toon shading (from left to right), and combines them into the final sketch (right).*

6.4 Keyframe Interpolation

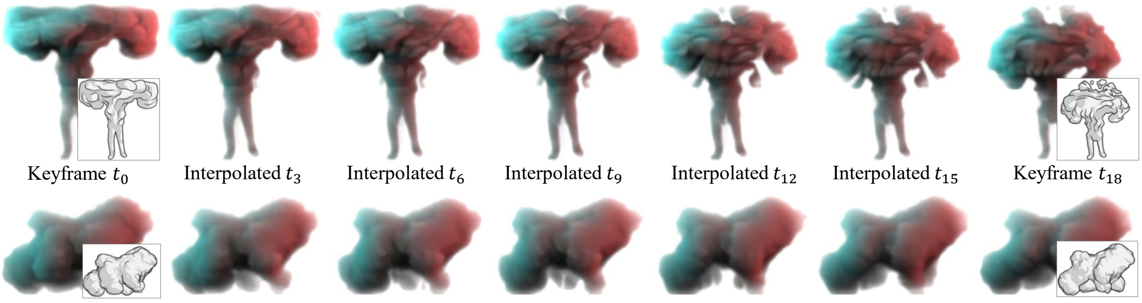


Figure 6.6: *From the keyframes reconstructed from sketches (left and right), we interpolate 5 intermediate frames with Wassertein barycenters to generate the inbetween fluid animation, for the artist scene* dissolving character *and the* cloudy puppy.

Although our main goal is to give control to the artist on the smoke density at prescribed keyframes, we also want to give an efficient preview on the look of the final, interpolated motion. One way to compute the interpolation between keyframes would be to compute a velocity field that advects the first keyframe into the next one, and to apply this iteratively over the sequence. We found that 1) the large difference of shape and total density between the generated keyframes makes this problem challenging to solve for state-of-the-art methods, such as the optical-flow optimization proposed by [Eckert et al., 2019] or the adjoint method [McNamara et al., 2004], and 2) it is not straightforward to interpolate a velocity field so that it conserves the trajectory of Semi-Lagrangian advection. Therefore, we used a velocity-free method, relying instead on optimal transport theory to compute directly any intermediate density field, while preserving the quality of the fluid motion.

One of the key advantages of such an approach is that we can compute the interpolated density at any given time of the simulation, without needing any information on the previous frames. As such, we select a time t between two keyframes at times t_0 and t_1, and we seek the interpolated density d between keyframes d_0 and d_1.

Wassertein barycenters are commonly used [Solomon et al., 2015] to interpolate between a family of probability distributions. Without loss of generality, we normalize the fluid densities so that their total sum is 1 and thus we assume that they represent a discrete probability distribution in 3D. In the following, we will therefore use the terms *density* and *distribution* interchangeably. In our setting, the keyframe interpolation problem writes as the Wassertein barycenter between d_0 and d_1:

$$\arg\min_{d} \quad (1-x)\,\mathcal{W}_2^2(d,d_0) + x\,\mathcal{W}_2^2(d,d_1), \tag{6.13}$$

where $\mathcal{W}_2^2$ is the squared 2-Wassertein distance, which denotes the cost of the optimal transport with a L2 norm, and $x = (t-t_0)/(t_1-t0)$ is the interpolation coefficient. Solving this problem requires using a slow linear programming algorithm, which is why we used the entropy-regularized variant [Cuturi, 2013] that introduces a Kernel $\mathcal{K}_\lambda(x,y) = e^{-(x-y)^2/\lambda}$ and shows that Equation 6.13 can be regularized as

$$\begin{aligned} \arg\min_{d} \quad & (1-x)\,\mathrm{KL}(\boldsymbol{\pi}_0|\mathcal{K}_\lambda) + x\,\mathrm{KL}(\boldsymbol{\pi}_1|\mathcal{K}_\lambda) \\ \text{s.t.} \quad & \boldsymbol{\pi}_0^\top \mathbf{1} = d_0 \text{ and } \boldsymbol{\pi}_1^\top \mathbf{1} = d_1 \\ & \boldsymbol{\pi}_0 \mathbf{1} = \boldsymbol{\pi}_1 \mathbf{1}, \end{aligned} \tag{6.14}$$

where KL is the Kullback-Leibler divergence, $\boldsymbol{\pi} \in \mathbb{R}^{3N} \times \mathbb{R}^{3N}$ is a transport plan between a pair of discrete 3D densities of side N, and $\mathbf{1}$ is the unit vector of $\mathbb{R}^{3N}$. Furthermore, each transport plan can be written as

$$\boldsymbol{\pi}_i = \mathrm{D}_{\mathbf{v}_i} \mathcal{K}_\lambda \mathrm{D}_{\mathbf{u}_i}, \tag{6.15}$$

where $D_{\mathbf{u}}$ and $D_{\mathbf{v}}$ are diagonal matrices which entries are filled with values of the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$. Previous work [Benamou et al., 2015] proposed to use *Bergman projections* to iteratively solve the problem, starting from $v = \mathbf{1}$ and progressively evaluating:

$$\begin{aligned} \mathbf{u}_i &\leftarrow d_i / \mathcal{K}_\lambda \mathbf{v}_i, \quad i \in [0,1] \\ \tilde{d} &= (\mathbf{v}_0 \mathcal{K}_\lambda \mathbf{u}_0)^{1-x}\ (\mathbf{v}_1 \mathcal{K}_\lambda \mathbf{u}_1)^{x} \\ \mathbf{v}_i &\leftarrow \tilde{d} / \mathcal{K}_\lambda \mathbf{u}_i, \quad i \in [0,1], \end{aligned} \tag{6.16}$$

where $\tilde{d}$ is the approximated barycenter.

Unfortunately this scheme does not scale well in practice, because the limit of the floating point numbers is quickly reached. [Schmitzer, 2019] proposes several ways to tackle this issue, one of which is to perform all the operations in log space, which is proved to converge for any value of λ. However, such an approach loose the separability of the kernel $\mathcal{K}_\lambda$, which is essential to keep a tractable running time, especially for 3D data. We propose an intermediate solution by introducing a log scaling function of the kernel:

$$\mathcal{S}_\lambda(\mathbf{x}) = \lambda \log \mathcal{K}_\lambda e^{\mathbf{x}/\lambda} \tag{6.17}$$

and log space variables $\boldsymbol{\alpha} = \lambda \log \mathbf{u}$ and $\boldsymbol{\beta} = \lambda \log \mathbf{v}$. We detail the process in Algorithm (4) and show interpolated frames in Figure 6.6.

Algorithm 4: *Stabilized Sinkhorn Iterations.*

```
$\beta \leftarrow \mathbf{1}$;
for n iterations do
  for $i \in [0,1]$ do
    $\boldsymbol{\alpha}_i \leftarrow \lambda \log d_0 - \mathcal{S}_\lambda(\boldsymbol{\beta}_i)$;
    $\gamma_i \leftarrow \boldsymbol{\beta}_i + \mathcal{S}_\lambda(\boldsymbol{\alpha}_i)$;
  end for
  $\boldsymbol{\mu} = (1-x)\,\gamma_0 + x\,\gamma_1$;
  for $i \in [0,1]$ do
    $\boldsymbol{\beta}_i = \boldsymbol{\mu} - \mathcal{S}_\lambda(\boldsymbol{\alpha}_i)$;
  end for
  return $e^{\boldsymbol{\mu}/\lambda}$;
end for
```

6.5 Training Data Generation

The generality of our dataset is key for a robust and high-quality reconstruction of diverse scenes sketched by artists, that can include both physically-inspired and non-physical shapes and motion of smoke. Therefore, we build a new dataset consisting of smoke densities, which captures a large variety of volumetric shapes. We also augment the output of our synthetic sketcher with several strategies to increase the robustness of our network with respect to small variations in sketching styles.

6.5.1 Simulation Taining Data

Our simulation dataset consists of $39,380$ density fields (from 1641 simulations) used for training and 600 density fields (25 simulations) for validation. Example snapshots from the training set are shown in Figure 6.8. Each simulation contains 20 frames without source and 30 frames with source computed at a resolution of 129^3. We run the simulation in *Houdini* with randomly changing temperature diffusion factor, cooling rate, viscosity, buoyancy strength and direction, as well as sharpening and turbulence factors.

For simulations without source, the pre-generated source is set as the initial density of the simulation. For the ones with source, we selected one of the regenerated sources and combined it with a max operator on the ongoing simulation. The number of sources for each simulation are [1,2,3,4] with probability [10,5,2,1]/18. The shapes of the initial sources are generated by computing a random shape from a set of union or difference of cube, cylinder and sphere, and setting the resulting shape as initial density of a fluid simulation that we run for 10 frames. We added this supplementary operation to remove sharp edges and corners found in the initial shapes and to capture a more diverse range of densities than the original binary ones.

Based on the density histogram, we find that both simulation strategies are complementary to each other. No source data are dominant by small density values, while with source data higher density values can be observed. In this way, our neural network model is not biased to learn to output small or large values only. It also helps to increase variation in the dataset so that we can handle various unseen data, such as simulation with and without sourcing.

6.5.2 Sketch Training Data and Augmentation

The synthetic sketcher allows us to inexpensively build a large dataset of smoke simulations coupled with sketches. We used sketch sizes of 258^2. To increase the robustness with respect to variations in artist style, we augmented the sketches used in the training to cover different values for brightness, contrast, contour strength, toon shading color, blurring and slurring [Simo-Serra et al., 2016] in x and y direction as noise addition, and variations in light direction in x and y. Figure 6.7 illustrates the different properties.

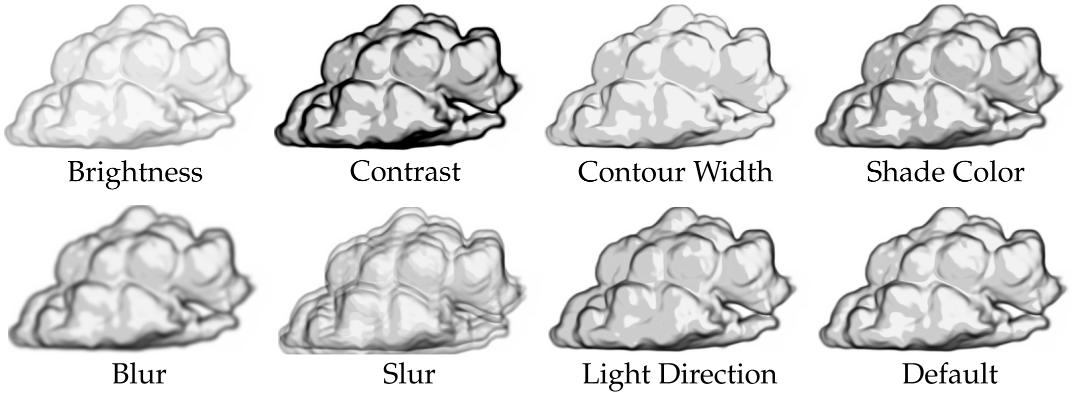


Figure 6.7: *Sketches are augmented to consider variations in sketching styles to increase robustness of the reconstruction.*

6.6 Results

6.6.1 Implementation and Performance

Sketch-to-Density CNN We implemented our system in *PyTorch* [Paszke et al., 2019] and used the Adam optimizer [Kingma and Ba, 2015] with a learning rate of 0.0001. Our network follows a U-Net architecture. In the decoder, we use nearest neighbor upsampling with flat 2D convolution instead of transposed convolution to avoid checkerboard artifacts [Odena et al., 2017]. The network is trained on density fields in the range $[-1, 1]$, using the full objective. In all our examples we used 3 passes in the training, and two subsequent refinement steps (front-left (fl)) at test time. Different settings are evaluated in the ablation study in Section 6.6.6.

Our model converges in 10 epochs, and the training time with our dataset takes up to 3 days using 4 NVIDIA GTX 1080 GPUs. At test time, the IVM computation with front and side input sketches takes $\sim$ 6.5ms per frame on a single GPU. Similarly, the update step of our trained updater CNN takes $\sim$ 3ms per frame. With refinements on front and left views (fl), the total inference time is $\sim$ 12.5ms per frame and hence our method is able to reconstruct 3D densities in real-time.

Keyframe Interpolation We found the proposed scheme to converge well even for small values of λ as $\lambda = 1$ when the average distance between the keyframes does not exceed about 30 voxels, which was the case in almost all our test scenes. In these cases, the iteration gave visual pleasing results after 15 iterations. To keep a low runtime, we separated the kernel as a 1D kernel, iteratively applied on each dimension of the volume, which leads to the overall complexity $O(kN^3)$, where $k < N$ is the footprint of the kernel and N the number of voxels in each dimension of the volume. For $N = 128$ and $\lambda = 1$, the maximum kernel size where the floating point evaluations are not null is 50. We also clamped the input of log functions to the smallest positive floating-point number to avoid numerical errors. Except for the 1D convolutions with a Gaussian kernel, all the operations can be parallelized. We implemented the algorithm on the GPU with *PyTorch*, where each iteration takes 0.3s and hence each frame 4.5s.

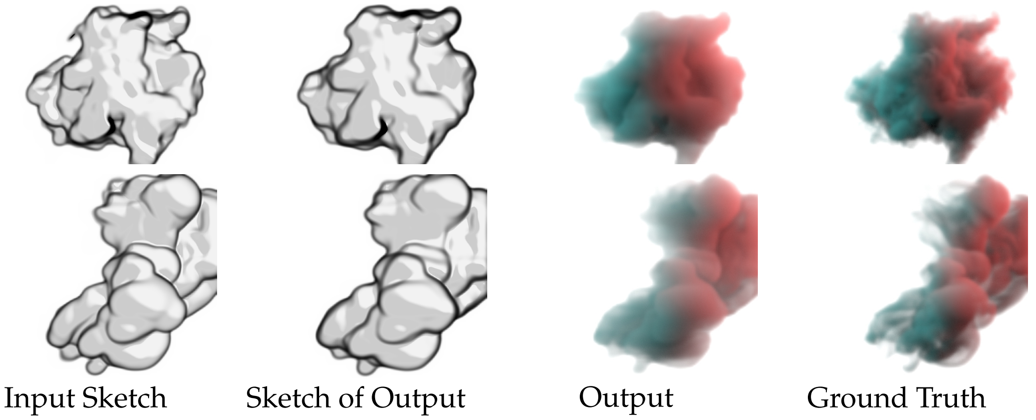


Figure 6.8: *Validation result with front view of seen (top) and unseen (bottom) examples of our dataset. From left to right: input sketch, sketch of reconstructed density, density after front-left refinement, ground truth density.*

6.6.2 Results on Our Dataset

Figure 6.8 shows the front view result of a reconstruction of the training dataset ('seen', top row) and validation set ('unseen', bottom row). From left to right we show the sketch of the ground truth, the input sketch, the reconstructed density, and the ground truth density. It can be seen that only minimal differences are visible between the sketches. The reconstructed density is smoother than the ground truth. Previous work [Biland et al., 2020] reported similar observations and claimed that lacking fine-scale details in smoke density reconstruction is a general problem of CNN-based flows. They discussed that adversarial losses (GAN) can hallucinate fine flow structures and improve visual plausibility. We found, however, that adding a discriminator loss to our network architecture is problematic as it degrades convergence. Alternatively, post-processing steps, such as turbulence synthesis [Kim et al., 2008b; Sato et al., 2018a] or deep upscaling [Xie et al., 2018], could be added to the reconstructions to counterbalance this problem.

6.6.3 Results on Synthetic Scenes

We show that our method generalizes well to various scenes by applying it to inputs from physically-based simulations, reconstructed 3D density fields from real smoke captures, and animation sequences including a character, cloud and puppy animation. For these examples, we generated synthetic sketch inputs by applying our sketcher to the given density field. It is important to note that the given density cannot be directly compared to the reconstructed density: our method infers a density such that its sketch is close to the input sketch.

Simulated Smoke

We used the smoke plume and smoke jet datasets of [Kim et al., 2019a] and applied our sketcher to the existing density field to generate sketch input keyframes (every 5th frame) for front and left views. Figure 6.9 shows the given density field, the synthetically generated sketch from this given density (input to our method), the sketch of the reconstructed density field, and the reconstructed density. The reconstructed density closely matches the footprint of the input sketch. We measure the quality of the inner details by comparing the synthetic sketches of the input with the one of our output, where we see that the shading and inner contours are well captured.

Captured Smoke Data

We applied our method to the *Scalarflow* dataset [Eckert et al., 2019] that contains reconstruction data of real-world captured smoke. We show the *ScalarFlow* density and its sketch, and the reconstruction results (sketch and density) for scene ID 1 in Figure 6.10. The sketches have no notion of smoke thickness and therefore cannot distinguish between a wispy and a dense smoke volume with identical sketch shapes. The sketch loss therefore has a tendency to favor reconstructions that are less wispy than the ones of *ScalarFlow*.

Animations

To evaluate the model on unseen smoke shapes, we used non-physical animation sequences as input and show the corresponding reconstructions in

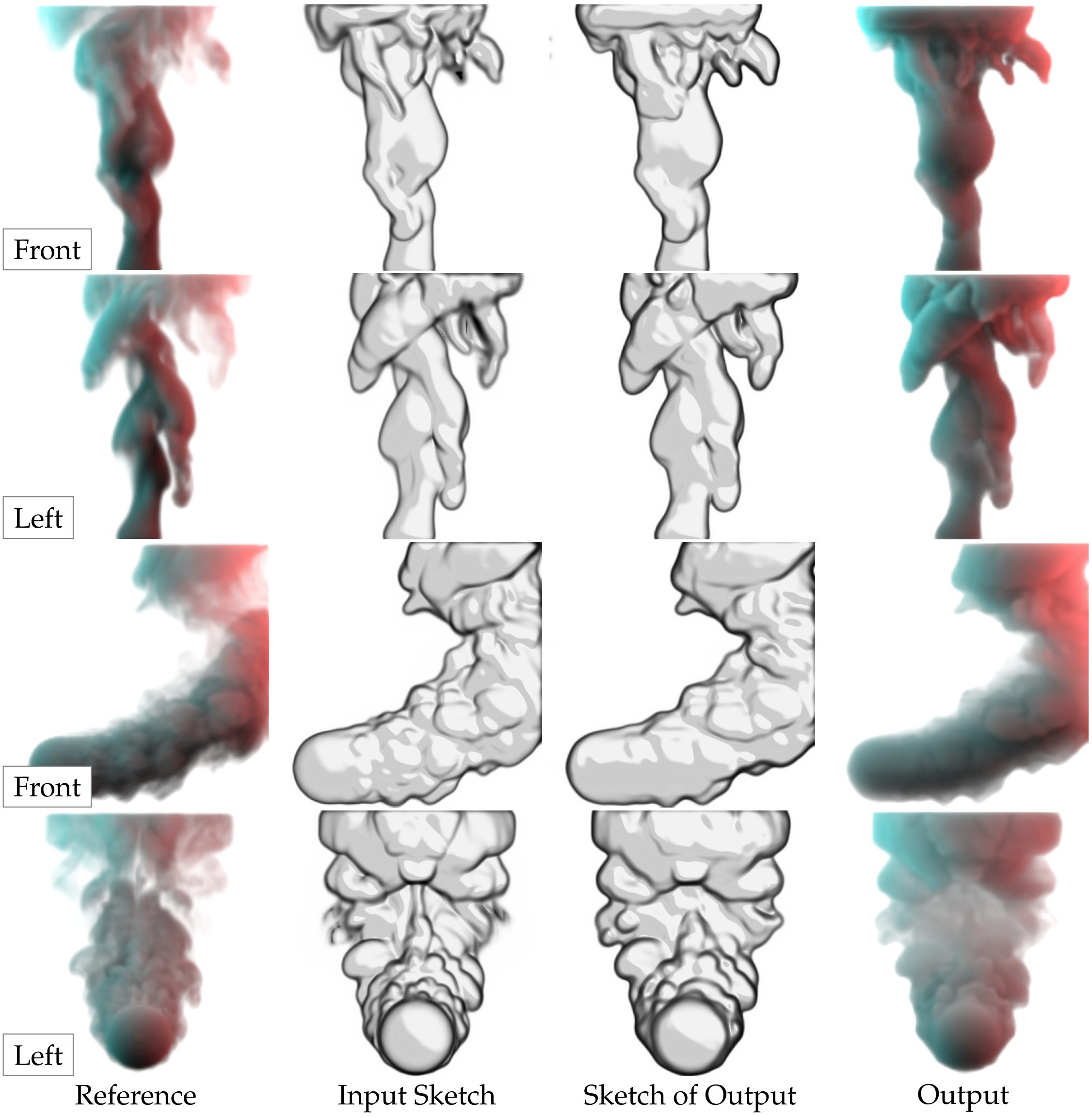


Figure 6.9: *Result using two physics-based simulation datasets of [Kim et al., 2019a] showing front and left views. From left to right: given density used for sketch generation, input sketch, sketch of the reconstructed density, reconstructed density.*

Figure 6.11. The top row show the results (front view) using a dancing character animation dataset from Adobe's *Mixamo Gallery*[1]. This data is not only difficult because no comparable examples were part of the training dataset, but also because of thin extremities and occlusions. We applied the method

[1] www.mixamo.com

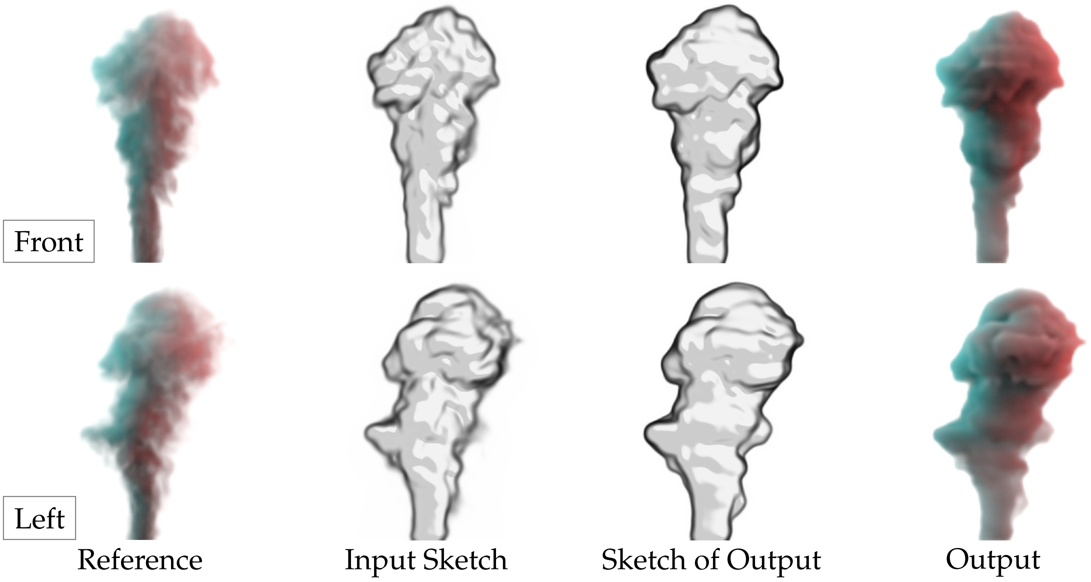


Figure 6.10: *Front (top) and left (bottom) views of a selected example from the* Scalarflow *dataset. From left to right:* ScalarFlow *density, input sketch, reconstructed sketch and output density.*

to a cloud[2], and a keyframe of a running puppy animation[3] is shown from side view in the last row. We also show more results in the accompanying video where we additionally amplified the character and puppy models with procedural clouds. For the character and puppy examples, we first converted the original meshes into volumes before computing the input sketches. Only minimal differences are visible when comparing the input sketch to the reconstruction sketch, demonstrating that our method is not only able to reliably reconstruct unseen shapes, but especially also handle non-physical (non-smoke) inputs.

Arbitrary Viewpoints

In most of our examples, we show results where the sketches are prescribed from canonical viewpoints (front and left views). We believe that this is one of the most complicated cases for density reconstruction because no information is shared between the different views. But we also show that our method can be applied efficiently to sketches from arbitrary viewpoints. The main - classical - issue is that the corners of the domain are cropped during rotation. We chose not to increase the size of the volume, but instead we

[2] www.technology.disneyanimation.com/clouds
[3] ©anonymous for review

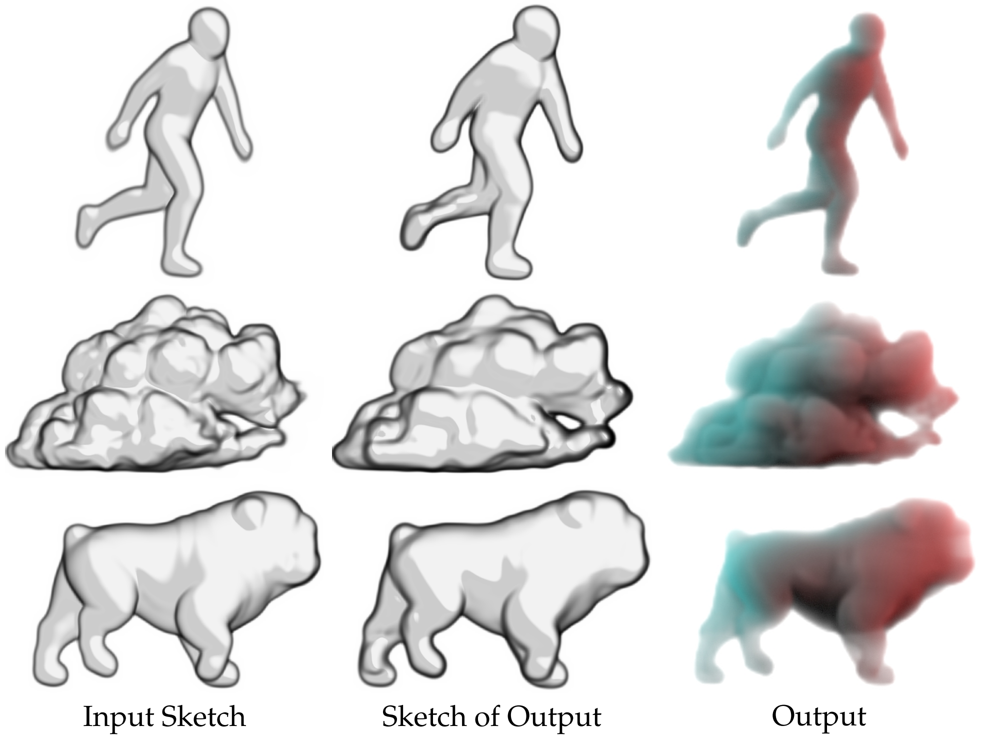


Figure 6.11: *Results of three unseen and non-physical animation sequences: dancing character (front view), clouds (front view), and running puppy (side view). From left to right: input sketch, sketch of reconstructed density, reconstructed density.*

reduce the domain to a sphere inscribed inside the 128^3 cube. A quick computation shows that proportionally less volume is wasted with this strategy. Figure 6.12 shows an example where the *puppy* model was reconstructed from sketches drawn from arbitrary viewpoints.

6.6.4 Results on Artist Sketches

We used sketches drawn by three different artists to evaluate our method at test time. We instructed the artists to sketch keyframes of the evolution of a smoke or an imaginary smoke-like object. Artists had to provide 10-20 keyframes for each scene, while the temporal distance inbetween keyframes not necessarily had to be uniform. The motion could disobey physical laws (cartoon animation, no volume conservation, inconsistencies with respect to shading in one as well as subsequent keyframes). We provided specific instructions with respect to image size (256x256), light direction (front-right, slightly up), shading (toon, white and grey), colors (grey-black lines, white background), and line width and style (up to 5px with Gaussian falloff).

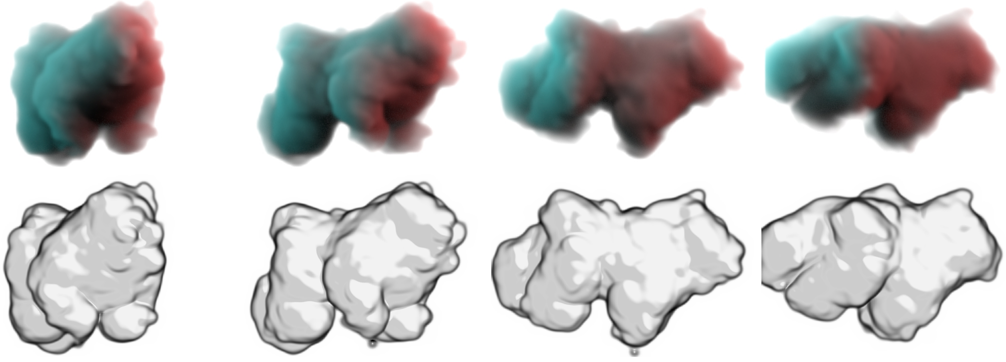


Figure 6.12: *Our method can seamlessly handle progressive refinement from arbitrary viewpoints despite training with canonical views only. The top and bottom rows show the refined density fields and input sketches, respectively.*

Keyframes of the first scene illustrate a character that transforms in a rising smoke cloud, which then sinks to the ground and dissolves. The second scene mimics a lion that transforms into a rhino while running. The last scene starts as a smoke explosion, the smoke then transforms into a bird-like object. Selected keyframes of the artist sequences are shown in Figure 6.1, and reconstruction results are depicted in Figure 6.13. Our method works surprisingly well for such highly non-physical examples. The reconstructed density and sketches thereof match quite closely the input sketches from the artists.

We evaluated the artists' experience with our method and their perception of its usefulness in production pipelines. The artists were very enthusiastic about the overall idea, the result they achieved, and the potential impact in industrial workflows. They especially liked the idea to use it as a tool to communicate early ideas with the client or art director, to clarify the specifications before moving to a heavier, more realistic simulation. The main drawback was the prescribed style, which had a small impact on the creativity and the learning curve. While this is a limitation of our method, all artists responded positively on the reliability and robustness of the reconstructions, which can be accounted to the prescribed style.

6.6.5 Keyframe Interpolation Result

We used the Wasserstein Barycenters method described in Section 6.4 to interpolate the final animation between the density keyframes reconstructed from the sketches. We evaluate the convergence of this approach by selecting resulting animations after several iterations, and we found that Sinkhorn's

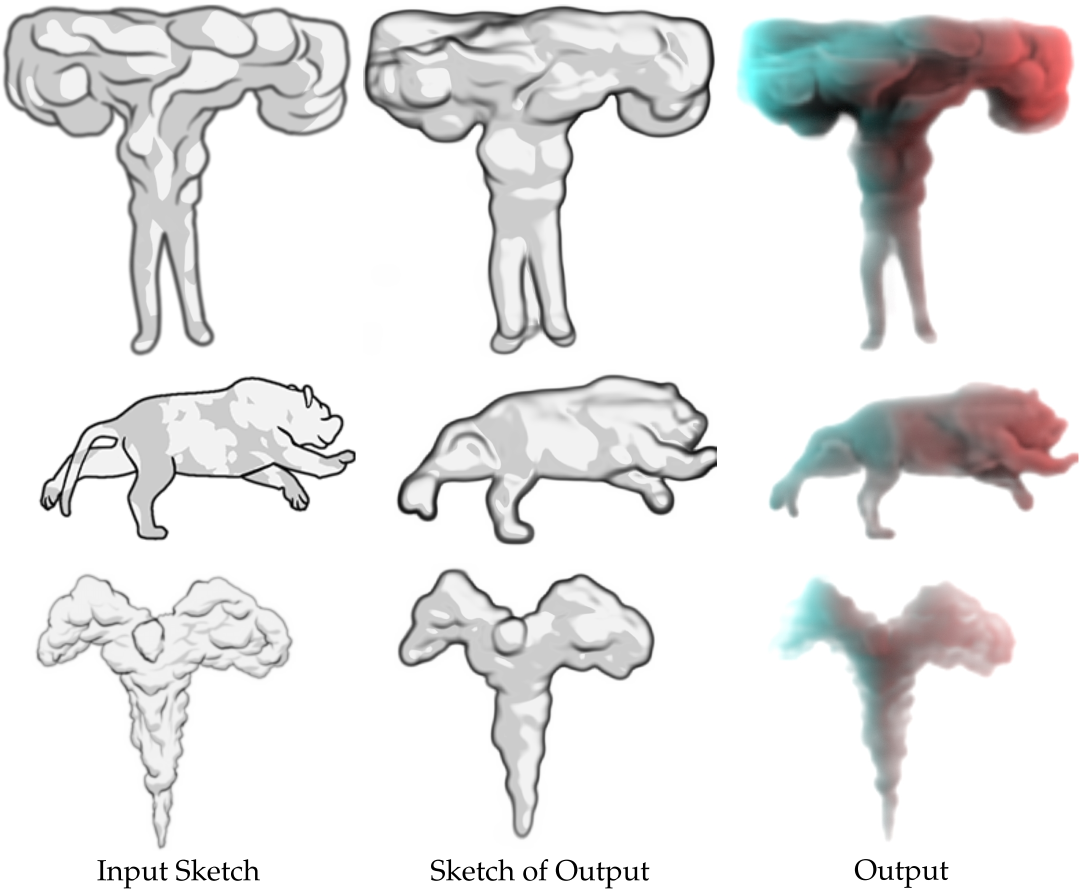


Figure 6.13: *Sketch inputs from three different artists (left) and reconstructed results (middle, right). From top to bottom:* dissolving character *(front view),* lion to rhino transform *(side view), and* bird *(front view).*

algorithm, even when not converged to the solution satisfying the optimal transport requirements, already results in a convincing motion after a few steps (we use $n = 15$ iterations in all examples). Furthermore, temporal consistency is preserved if the number of iterations is the same for subsequent frames. A sample of interpolated frames is shown in Figure 6.6, between two keyframes of the *artist* scene, and between reconstructions of a cloudy variant of the *puppy*.

The computation cost of 300ms per iteration might seem at first sight prohibitive for an interactive design tool, but the approach has several advantages that makes it competitive: 1) the first iterations already result in a convincing animation. This enables the implementation of a *preview* software, where the frames are refined in the background while the user previews the results. 2) As opposed to methods based on simulations, where the user has to wait for the evaluation of the whole sequence before obtaining the result, the computation of one frame of our method is independent of the previous ones. It is also possible to generate all the frames in parallel on a corresponding architecture. 3) The interpolation is continuous in time and therefore compatible with any temporal deformation. 4) We found that none of the previous work that we implemented, among adjoint method [McNamara et al., 2004] or optical-flow optimization [Eckert et al., 2019], were able to converge when provided with pairs of our density keyframes, because of the sometimes drastic difference in shape and total density.

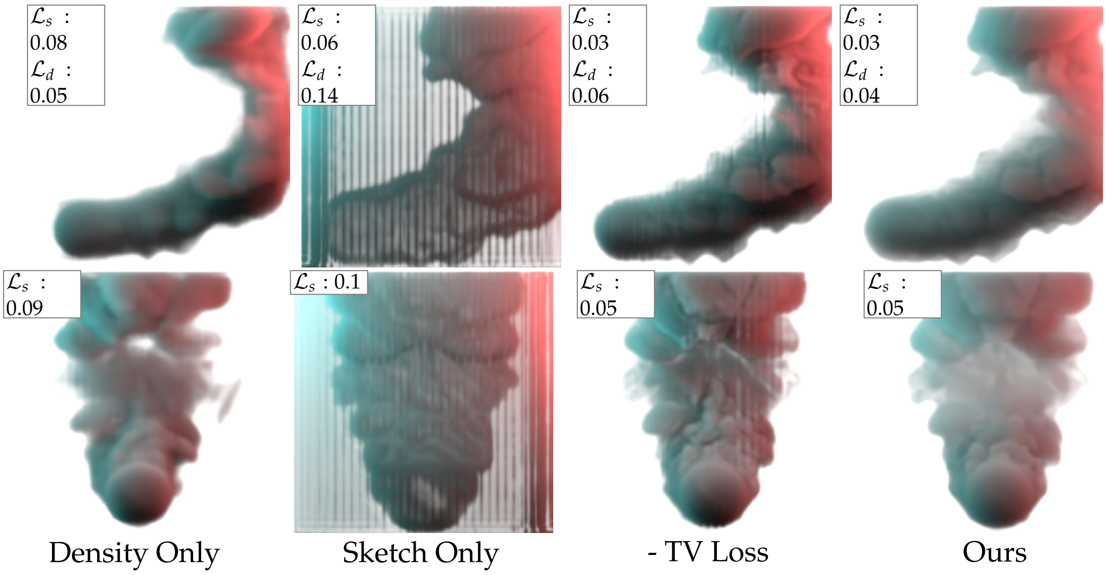

$\mathcal{L}_s$: 0.08 $\mathcal{L}_d$: 0.05
$\mathcal{L}_s$: 0.06 $\mathcal{L}_d$: 0.14
$\mathcal{L}_s$: 0.03 $\mathcal{L}_d$: 0.06
$\mathcal{L}_s$: 0.03 $\mathcal{L}_d$: 0.04
$\mathcal{L}_s$: 0.09
$\mathcal{L}_s$: 0.1
$\mathcal{L}_s$: 0.05
$\mathcal{L}_s$: 0.05
Density Only
Sketch Only
- TV Loss
Ours

Figure 6.14: *Evaluation of loss functions. From left to right: density loss, sketch loss, density+sketch losses, density+sketch+depth variation losses (ours)*

6.6.6 Ablation Study

Loss Functions

We evaluated the impact of the different loss functions and illustrate the results in Figure 6.14. Using only the density loss (Equation (6.1)) like in previous work [Delanoy et al., 2018] we observe large discrepancies between the input sketch and the sketch of the reconstructed density. When using only the sketch loss (Equation (6.2)) results are very detailed but at the cost of depth ambiguity. If both density and sketch losses are used, the sketch correspondence as well as the depth reconstruction are improved, but noise is well visible. Adding the total variation loss in depth direction (Equation (6.3)) eliminates these artifacts and generates the best results (our model). Figure 6.14 also embeds a quantitative quality measure (value of density and sketch loss). Lower values stand for higher accuracy.

Recursive Passes

We evaluate our network with various numbers of passes at training and test time. We show the results on front view only in Figure 6.15, where each line corresponds to 1, 2 and 3 passes during training, respectively, and each column to 1, 3 and 6 successive refinements at test time that we denote by 'f', 'fff' and 'ffffff'. We observed that the difference between training pass 3 and 4 is marginal; we therefore used 3 passes in all our examples. They result in a clearly better reconstruction than the single pass version, which justifies the use of our rollback loss.

On the other hand, with 3 and more passes during training, the number of successive refinements at test time have less influence. The results in 'f' already recovered most of the features of the input sketches, and is slightly improved by 'fff', which is hardly distinguishable from 'ffffff'. Next, we alternate front and left view refinement steps (f, fl, flf, flfl) and show the impact on the reconstruction quality in Figure 6.3. For most exmaples, a front view optimization followed by a left view refinement (fl) is sufficient; for some examples (i.e., simulated smoke) quality was further improved by additional refinement steps (flfl).

Sketch Augmentation

We argued that using augmented sketches in the training favors generality. We evaluate this property in Figure 6.16 and show comparisons of reconstructions from different sketches modified to be far from our initial parameters. On the top row, we input our synthetic sketch of the character scene, where we reduced the brightness and the line width. On the bottom row, we use one of the artist examples that is far from the prescribed style. The middle column shows our results with data augmentation, while the right one shows the results without. Several parts of the sketches were not reconstructed, which shows the need for data augmentation for more generality on the sketch style.

Different Dataset

We justify the need for a new dataset by training our model with the state-of-the art *Scalarflow* dataset [Eckert et al., 2019] captured from real smoke. We show in Figure 6.17 our results with the character scene (top), and the ones obtained after training with *ScalarFlow* (bottom). The results indicate that although *ScalarFlow* exhibits smokes of extremely good quality, it does not embed enough variability to reconstruct arbitrary scenes.

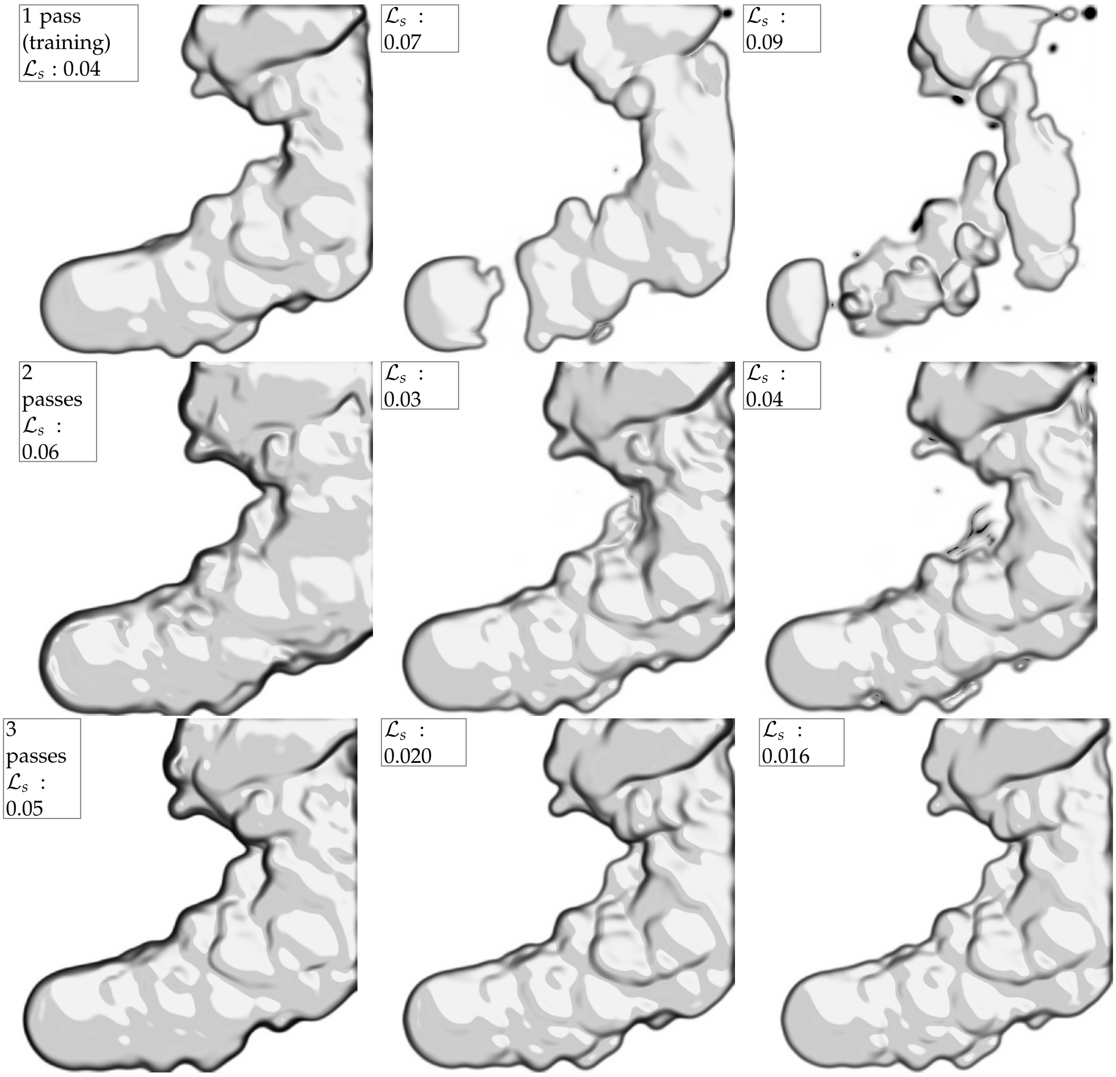


Figure 6.15: *Evaluation of recursive passes (during training) and inference sequence (at test time). We use 3 passes in practice.*

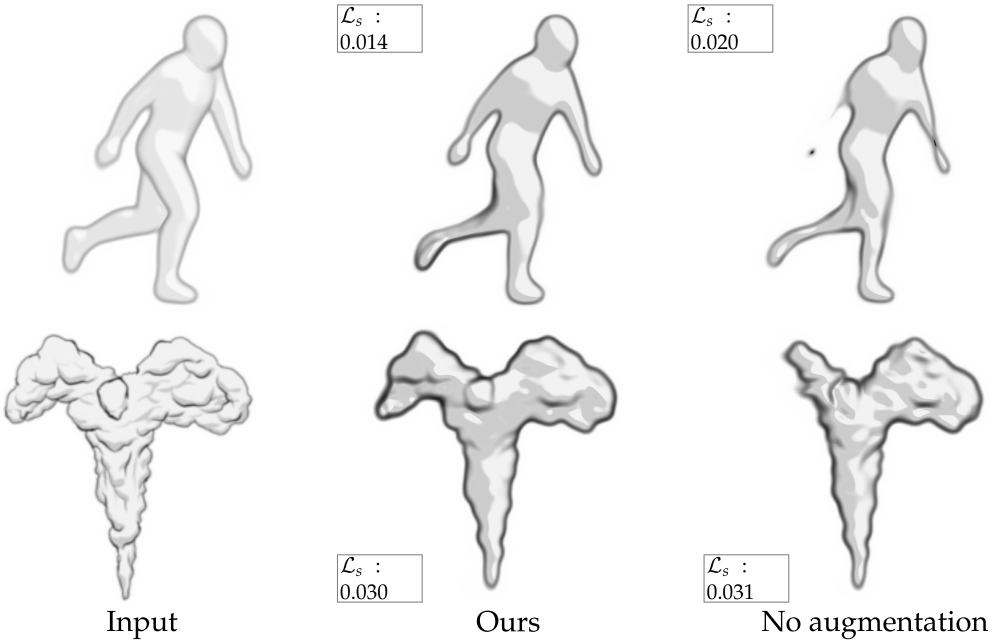


Figure 6.16: *Using sketch variations in the training improves robustness of the reconstruction quality. We increased the brightness and reduced the size of the contour of the sketch of the* character *example (top row), and tested with the* bird *scene (bottom row). Left is the input, middle column our results after training with sketch augmentation, and right results without augmentation.*

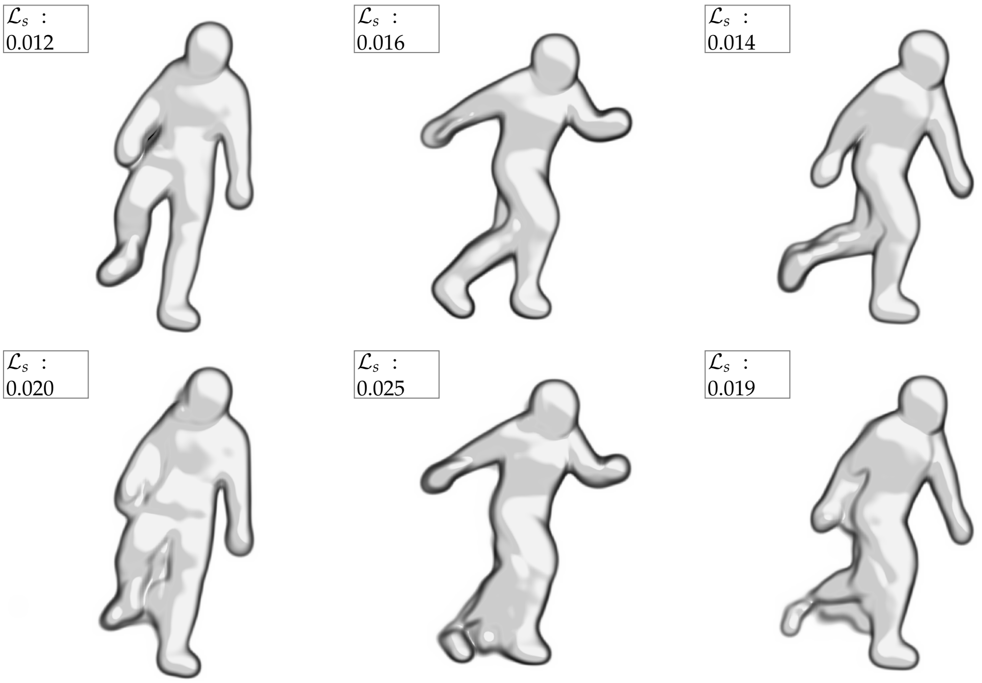


Figure 6.17: *3D reconstructions computed with a model trained on our dataset (top) and* ScalarFlow *(bottom), highlighting the importance of using a well designed training dataset for generic application.*

CHAPTER 7

Conclusion

In this thesis, we have proposed novel algorithms that allow better integration of fluid simulations into artists' workflow, mainly focusing on computational efficiency and artistic controllability without losing its physical plausibility and stability. The technical foundation is based on data-driven concepts: we leveraged machine learning to improve existing simulations and enabled innovative applications.

7.1 Principal Contributions

In Chapter 3, we have presented the first generative deep learning architecture, *Deep Fluids*, that successfully synthesizes plausible and divergence-free 2D and 3D fluid simulation velocities from a set of reduced parameters. Our results show that generative neural networks are able to construct a wide variety of fluid behaviors, from turbulent smoke to viscous liquids, that closely match the input training data. Moreover, our network can synthesize physically plausible motion when the input parameters are continuously sampled from intermediate states not present during training. In addition, we can handle complex parameterizations by using a novel Latent Space Integration (LSI) network. Our solver has a constant evaluation time and is considerably faster (up to 700×) than simulations with the underlying CPU solver, which makes our approach attractive for re-simulation scenarios where input interactions can be parameterized. These performance characteristics immediately suggest applications in games and virtual environments. As high-resolution fluid simulations are also known to demand large disk and memory budgets, the compression characteristics of our algorithm (with up to 1300×) render the method attractive for movie productions as well. Our CNN architecture was carefully designed to achieve high quality fluid simulations, which is why the loss function considers both the velocity field and its gradient.

In Chapter 4, we have presented *TNST*, the first Transport-based Neural Style Transfer algorithm for smoke simulations. Our method facilitates automatic instantiation of a vast set of motifs through semantic transfer, enabling novel artistic manipulations for fluid simulation data. Additionally, the proposed method successfully synthesizes various styles of input images, ranging from artistic to photorealistic examples. Even though 2D CNNs are employed, our differentiable renderer allows the creation of 3D volumetric structures from a small set of views. Our stylization algorithm is able to handle high-resolution simulations up to 16 million voxels.

In Chapter 5, we have presented *LNST*, a Lagrangian approach for neural flow stylization and have demonstrated benefits with respect to quality (improved temporal coherence), performance (stylization per frame in less than a minute), and art-directability (multi-fluid stylization, color transfer, liquid stylization). A key property of our approach is that it is not restricted to any particular fluid solver type (i.e., grids, particles, hybrid solvers). To enable this, we have introduced a strategy for grid-to-particle transfer (and vice versa) to efficiently update attributes and gradients, and a re-simulation that can be effectively applied to grid and particle fluid representations. This generality of our method facilitates seamless integration of neural style transfer into existing content production workflows.

In Chapter 6, we have presented a method for *reconstructing 3D smoke densities from 2D artist sketches*, which potentially represents the first step towards bridging the gap between early-stage prototyping of smoke keyframes and visual realization. We proposed a CNN for computing density refinements, a differentiable sketch renderer integrated into the end-to-end training, and a set of loss functions designed explicitly for the sketch-to-density problem. The training dataset was carefully designed for general applicability, demonstrated by applying the method to diverse datasets ranging from physics simulations, captured real-world flows, procedural clouds, character animation, and highly non-physical artist sketches.

7.2 Future Work

Overall, we found that the proposed Generative CNNs are able to produce density and velocity fields accurately. However, for small-scale details or near discontinuities such as boundary conditions, the network can sometimes smooth out fine flow structures. A possible future research direction is the exploration of generative adversarial networks (GANs), partial convolutions [Liu et al., 2018a], joint training with other fields such as SDF, or alternative distance measures to enhance the accuracy for delicate structures in the data. Additionally, it is an interesting direction to improve the Latent Space Integration network in various ways, such as using an end-to-end training with the autoencoder or the gradient loss for temporal smoothness.

We are not aware of any other methods that use a volumetric differentiable rendering for optimizing 3D smoke data, and we believe that our work may inspire further research in this direction. For example, our differentiable renderer could be employed for reconstructing 3D smoke volumes from images as in [Eckert et al., 2018] and be extended to transfer image-based filters as in [Liu et al., 2018b]. Similar to the smoke renderer, a dedicated differentiable renderer for liquids would improve the resulting quality and especially also support a wider range of liquid simulation setups. As further extensions, super-resolution can be thought of as a specific type of style transfer [Johnson et al., 2016], and we believe that our work can be tailored towards improving current super-resolution methods for fluids [Xie et al., 2018].

We have shown that TNST and LNST enable novel effects and a high degree of art-directability, rendering flow stylization more practical in production workflows. However, we have not tested the method on large-scale simulations that are typically used in such settings. While our LNST method can handle up to 2 million particles, larger scenes are restricted by the available memory. Moreover, in practical settings, the scene complexity is higher, potentially posing challenges concerning artist control of the stylization. Reducing the computation time for stylizing an entire simulation from one day with TNST to a single hour with LNST renders the method much more practical for digital artists. However, for testing different input structures, a real-time method would be desirable. Recent concepts presented on neural image stylization might be mapped to 3D simulations to improve efficiency further.

The main limitation of our sketch-to-density pipeline is that we cannot handle arbitrary drawing styles of artists. We rely on the fact that artists follow our design principles for the sketching style, which is, for example, using clean contours and two-color toon shading. We used sketch augmentation during training, though, to increase robustness with respect to smaller variations in the sketch style, such as using different contour widths, brightness, or light direction. Future work requires extensive user studies with artists to evaluate the design choices of our differentiable sketch generator in real settings, as well as future research to support a wider variety of sketch styles while maintaining reliable and robust reconstructions.

Designing entire physically-based animations by a few input sketches could potentially have a massive impact, as this would not only simplify authoring processes but also make physics simulations accessible to novice users without requiring any knowledge base in fluid modeling and solver parameters. However, a major difficulty is to compute physically-plausible transitions (in-between keyframes) between two reconstructed density keyframes. This is especially challenging as we wanted to give an artist as much freedom as possible and support non-realistic shapes and highly non-physical motion. We found that previous methods for keyframe interpolation could not handle this, and therefore we developed an interpolation method using Wasserstein barycenters. During the algorithm design, we had in mind that the method must be efficient at test time to enable interactive prototyping, authoring, and pre-visualization of smoke animations. This is a pre-requisite for future integration into workflows and tools used by artists. Our total inference time per frame is 12.5ms and hence enables real-time previews during sketching. The interpolation is with 4.5s per frame more expensive, but still fast and could run continuously in the background to provide instant previews of the resulting animation.

To conclude, we believe machine learning for fluid simulations still has a huge amount of potential with unexplored areas, and we hope this thesis has contributed to another step forward in this field.

References

[Abadi et al., 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2016.

[Ando and Tsuruno, 2011] Ryoichi Ando and Reiji Tsuruno. A particle-based method for preserving fluid sheets. In *Proceedings - SCA 2011: ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 7–16, New York, New York, USA, 2011. ACM Press.

[Ando et al., 2013] Ryoichi Ando, Nils Thürey, and Chris Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics*, 32(4):1, 7 2013.

[Ando et al., 2015a] Ryoichi Ando, Nils Thuerey, and Chris Wojtan. A stream function solver for liquid simulations. *ACM Transactions on Graphics*, 34(4):53:1–53:9, 2015.

[Ando et al., 2015b] Ryoichi Ando, Nils Thürey, and Chris Wojtan. A Dimension-reduced Pressure Solver for Liquid Simulations. *Computer Graphics Forum*, 34(2):473–480, 5 2015.

[Angelidis et al., 2006] Alexis Angelidis, Fabrice Neyret, Karan Singh, and Derek Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *Computer Animation, Conference Proceedings*, volume 02-04-Sept of *SCA '06*, pages 25–32, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Azevedo and Oliveira, 2013] Vinicius C. Azevedo and Manuel M. Oliveira. Efficient smoke simulation on curvilinear grids. *Computer Graphics Forum*, 32(7):235–244, 10 2013.

[Barbǐ and Popoví, 2008] Jernej Barbǐ and Jovan Popoví. Real-time control of physically based simulations using gentle forces. *ACM Transactions on Graphics*, 27(5):1–10, 12 2008.

[Bargteil et al., 2006] Adam W. Bargteil, Funshing Sin, Jonathan E. Michaels, Tolga G. Goktekin, and James F. O'Brien. A texture synthesis method for liquid animations. In *Computer Animation, Conference Proceedings*, volume 02-04-Sept of *SCA '06*, pages 345–351, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Barnes and Zhang, 2017] Connelly Barnes and Fang Lue Zhang. A survey of the state-of-the-art in patch-based synthesis. *Computational Visual Media*, 3(1):3–20, 3 2017.

[Becker and Teschner, 2007] Markus Becker and Matthias Teschner. Weakly compressible SPH for free surface flows. Technical report, 2007.

[Benamou et al., 2015] Jean David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyŕ. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

[Bender and Koschier, 2015] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings - SCA 2015: 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 147–155, New York, New York, USA, 2015. ACM Press.

[Bender, 2016] Jan Bender. SPlisHSPlasH, 2016.

[Bi et al., 2017] Sai Bi, Nima Khademi Kalantari, and Ravi Ramamoorthi. Patch-based optimization for image-based texture mapping. *ACM Transactions on Graphics*, 36(4):1–11, 7 2017.

[Biland et al., 2020] Simon Biland, Vinicius C. Azevedo, Byungsoo Kim, and Barbara Solenthaler. Frequency-Aware Reconstruction of Fluid Simulations with Generative Networks. In *Eurographics Short Paper 2020*, 2020.

[Bodin et al., 2012] Kenneth Bodin, Claude Lacoursière, and Martin Servin. Constraint fluids. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):516–526, 2012.

[Bousseau et al., 2007] Adrien Bousseau, Fabrice Neyret, Joëlle Thollot, and David Salesin. Video watercolorization using bidirectional texture advection. *ACM Transactions on Graphics*, 26(3):104, 7 2007.

[Brackbill et al., 1988] J. U. Brackbill, D. B. Kothe, and H. M. Ruppel. Flip: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38, 1 1988.

[Bridson, 2007] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches, SIGGRAPH'07*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Bridson, 2015] Robert Bridson. *Fluid simulation for computer graphics*. A K Peters/CRC Press, 9 2015.

[Browning et al., 2014] Mark Browning, Connelly Barnes, Samantha Ritter, and Adam Finkelstein. Stylized keyframe animation of fluid simulations. In *NPAR Symposium on Non-Photorealistic Animation and Rendering*, volume 2014-Janua, pages 63–70. ACM, 2014.

[Chorin, 1967] Alexandre Joel Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, 8 1967.

[Chorin, 1969] Alexandre Joel Chorin. On the convergence of discrete approximations to the Navier-Stokes equations. *Mathematics of Computation*, 23(106):341–341, 5 1969.

[Christen et al., 2020] Fabienne Christen, Byungsoo Kim, Vinicius C. Azevedo, and Barbara Solenthaler. Neural Smoke Stylization with Color Transfer. In *Eurographics 2020 Short Paper*, 2020.

[Chu and Thuerey, 2017] Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics*, 36(4):1–14, 7 2017.

[Cordier et al., 2016] Frederic Cordier, Karan Singh, Yotam Gingold, and Marie-Paule Cani. Sketch-Based Modeling. In *SIGGRAPH ASIA 2016 Courses*, SA '16. Association for Computing Machinery, 2016.

[Cui et al., 2018] Qiaodong Cui, Pradeep Sen, and Theodore Kim. Scalable laplacian eigenfluids. *ACM Transactions on Graphics*, 37(4):1–12, 8 2018.

[Cuturi, 2013] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, pages 2292–2300, 2013.

[De Witt et al., 2012] Tyler De Witt, Christian Lessig, and Eugene Fiume. Fluid simulation using Laplacian eigenfunctions. *ACM Transactions on Graphics*, 31(1):1–11, 2012.

[DeCarlo et al., 2003] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM SIGGRAPH 2003 Papers, SIGGRAPH '03*, 22(3):848–855, 2003.

[Delanoy et al., 2018] Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A. Efros, and Adrien Bousseau. 3D Sketching using Multi-View Deep Volumetric Prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(1):1–22, 2018.

[Demby Jones et al., 2016] Aaron Demby Jones, Pradeep Sen, and Theodore Kim. Compressing Fluid Subspaces. In *SCA '16: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16, pages 77–84, Aire-la-Ville, Switzerland, Switzerland, 2016. Eurographics Association.

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

[Diamanti et al., 2015] Olga Diamanti, Connelly Barnes, Sylvain Paris, Eli Shechtman, and Olga Sorkine-Hornung. Synthesis of complex image appearance from limited exemplars. *ACM Transactions on Graphics*, 34(2):1–14, 3 2015.

[Eckert et al., 2018] M. L. Eckert, W. Heidrich, and N. Thuerey. Coupled Fluid Density and Motion from Single Views. *Computer Graphics Forum*, 37(8):47–58, 6 2018.

[Eckert et al., 2019] Marie Lena Eckert, Kiwon Um, and Nils Thuerey. ScalarFlow: A large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics*, 38(6), 2019.

[Enright et al., 2002] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02*, pages 736–744, New York, New York, USA, 2002. ACM Press.

[Farimani et al., 2017] Amir Barati Farimani, Joseph Gomes, and Vijay S. Pande. Deep Learning the Physics of Transport Phenomena. 2017.

[Fattal and Lischinski, 2004] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. In *ACM SIGGRAPH 2004 Papers, SIGGRAPH 2004*, pages 441–448, New York, New York, USA, 2004. ACM Press.

[Fedkiw et al., 2001] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001*, pages 15–22, New York, New York, USA, 2001. ACM Press.

[Ferstl et al., 2016] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 5 2016.

[Fišer et al., 2016] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. StyLit: Illumination-guided example-based stylization of 3D renderings. *ACM Transactions on Graphics*, 35(4):1–11, 2016.

[Flynn et al., 2019] Sean Flynn, Parris Egbert, Seth Holladay, and Bryan Morse. Fluid carving: Intelligent resizing for fluid simulation data. *ACM Transactions on Graphics*, 38(6):1–14, 2019.

[Fong et al., 2017] Julian Fong, Magnus Wrenninge, Christopher Kulla, and Ralf Habel. Production volume rendering SIGGRAPH 2017 course. In *ACM SIGGRAPH 2017 Courses, SIGGRAPH 2017*, pages 1–79, New York, New York, USA, 2017. ACM Press.

[Foster and Metaxas, 1996] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 9 1996.

[Fu et al., 2017] Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. A polynomial particle-in-cell method. *ACM Transactions on Graphics*, 36(6):1–12, 11 2017.

[Gagnon et al., 2016] Jonathan Gagnon, François Dagenais, and Eric Paquette. Dynamic lapped texture for fluid simulations. *Visual Computer*, 32(6-8):901–909, 6 2016.

[Gagnon et al., 2019] Jonathan Gagnon, Julián E. Guzmán, Valentin Vervondel, François Dagenais, David Mould, and Eric Paquette. Distribution Update of Deformable Patches for Texture Synthesis on the Free Surface of Fluids. *Computer Graphics Forum*, 38(7):491–500, 10 2019.

[Gatys et al., 2016a] Leon Gatys, Alexander Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *Journal of Vision*, 16(12):326, 2016.

[Gatys et al., 2016b] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 2414–2423. IEEE, 6 2016.

[Gerszewski et al., 2015] Dan Gerszewski, Ladislav Kavan, Peter Pike Sloan, and Adam W. Bargteil. Basis enrichment and solid-fluid coupling for model-reduced fluid simulation. *Computer Animation and Virtual Worlds*, 26(2):109–117, 2015.

[gfv, 2015] GoogleNet Feature Visualizations (http://storage.googleapis.com/deepdream/visualz 2015.

[Gilland, 2012] Joseph Gilland. *Elemental Magic Volume II – The Technique of Special Effects Animation*. CRC Press, 2012.

[Gingold and Monaghan, 1977] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389, 12 1977.

[Gissler et al., 2019] Christoph Gissler, Andreas Peer, Stefan Band, Jan Bender, and Matthias Teschner. Interlinked SPH pressure solvers for strong fluid-rigid coupling. *ACM Transactions on Graphics*, 38(1):1–5, 2019.

[Goodfellow et al., 2014] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 2672–2680. Curran Associates, Inc., 2014.

[Guérin et al., 2017] Éric Guérin, Julie Digne, Éric Galin, Adrien Peytavie, Christian Wolf, Bedrich Benes, and Benoît Martinez. Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM Transactions on Graphics*, 36(6), 2017.

[Guo et al., 2016] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-Augu, pages 481–490, New York, New York, USA, 2016. ACM Press.

[Gupta and Narasimhan, 2007] Mohit Gupta and Srinivasa G. Narasimhan. Legendre fluids: A unified framework for analytic reduced space modeling and rendering of participating media. In *Symposium on Computer Animation 2007 - ACM SIGGRAPH / Eurographics Symposium Proceedings, SCA 2007*, pages 17–26, 2007.

[Han et al., 2017] Xiaoguang Han, Chang Gao, and Yizhou Yu. DeepSketch2Face: A deep learning based sketching system for 3D face and caricature modeling. *ACM Transactions on Graphics*, 36(4), 2017.

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

[He et al., 2012] Xiaowei He, Ning Liu, Sheng Li, Hongan Wang, and Guoping Wang. Local poisson SPH for viscous incompressible fluids. *Computer Graphics Forum*, 31(6):1948–1958, 2012.

[He et al., 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778, 2016.

[Holl et al., 2020] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to Control PDEs with Differentiable Physics. In *International Conference on Learning Representations*, 2020.

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

[Hu et al., 2019a] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. DiffTaichi: Differentiable Programming for Physical Simulation. In *ICLR*, 2019.

[Hu et al., 2019b] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B. Tenenbaum, William T. Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. ChainQueen: A real-time differentiable physical simulator for soft robotics. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2019-May, pages 6265–6271, 2019.

[Hu et al., 2019c] Yuanming Hu, Xinxin Zhang, Ming Gao, and Chenfanfu Jiang. On hybrid lagrangian-eulerian simulation methods: Practical notes and high-performance aspects. In *ACM SIGGRAPH 2019 Courses, SIGGRAPH 2019*, page 16. ACM, 2019.

[Huang et al., 2017] Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(8):2003–2013, 2017.

[Ihmsen et al., 2012] Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. Unified spray, foam and air bubbles for particle-based fluids. *Visual Computer*, 28(6-8):669–677, 2012.

[Ihmsen et al., 2014] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):426–435, 2014.

[Inglis et al., 2017] T. Inglis, M. L. Eckert, J. Gregson, and N. Thuerey. Primal-Dual Optimization for Fluids. *Computer Graphics Forum*, 36(8):354–368, 2017.

[Insafutdinov and Dosovitskiy, 2018] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 2802–2812, 2018.

[Isola et al., 2017] Phillip Isola, Jun Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:5967–5976, 2017.

[Jaderberg et al., 2015] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *Advances in Neural Information Processing Systems*, 2015-Janua:2017–2025, 6 2015.

[Jamriška et al., 2015] Ondřej Jamriška, Jakub Fišer, Paul Asente, Jingwan Lu, Eli Shechtman, and Daniel Sýkora. LazyFluids: Appearance transfer for fluid animations. *ACM Transactions on Graphics*, 34(4):92, 2015.

[Jiang et al., 2015] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine Particle-In-Cell method. *ACM Transactions on Graphics*, 34(4):1–51, 7 2015.

[Jiang et al., 2016] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses, SIGGRAPH 2016*, pages 1–52. 2016.

[Jing et al., 2019] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural Style Transfer: A Review. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2019.

[Johnson et al., 2016] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *European Conference on Computer Vision 2016*, volume 9906 LNCS, pages 694–711. Springer, 3 2016.

[Kallweit et al., 2017] Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics*, 36(6), 2017.

[Karras et al., 2018] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[Kato et al., 2018] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D Mesh Renderer. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018.

[Kim and Delaney, 2013] Theodore Kim and John Delaney. Subspace fluid re-simulation. *ACM Transactions on Graphics*, 32(4):1, 7 2013.

[Kim et al., 2005] Byung Moon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. FlowFixer: Using BFECC for fluid simulation. In *Natural Phenomena*, NPH'05, pages 51–56, Goslar, DEU, 2005. Eurographics Association.

[Kim et al., 2006a] Janghee Kim, Deukhyun Cha, Byungjoon Chang, Bonki Koo, and Insung Ihm. Practical animation of turbulent splashing water. In *Computer Animation, Conference Proceedings*, volume 02-04-Sept of *SCA '06*, pages 335–344, Goslar, DEU, 2006. Eurographics Association.

[Kim et al., 2006b] Yootai Kim, Raghu MacHiraju, and David Thompson. Path-based control of smoke simulations. In *Computer Animation, Conference Proceedings*, volume 02-04-Sept of *SCA '06*, pages 33–42. Eurographics Association, 2006.

[Kim et al., 2008a] Doyub Kim, Oh Young Song, and Hyeong Seok Ko. A semi-lagrangian CIP fluid solver without dimensional splitting. *Computer Graphics Forum*, 27(2):467–475, 4 2008.

[Kim et al., 2008b] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. In *ACM SIGGRAPH 2008 papers on - SIGGRAPH '08*, volume 27, page 1, New York, New York, USA, 2008. ACM, ACM Press.

[Kim et al., 2013] Theodore Kim, Jerry Tessendorf, and Nils Thürey. Closest Point Turbulence for liquid surfaces. *ACM Transactions on Graphics*, 32(2):15, 2013.

[Kim et al., 2019a] Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2019)*, 38(6):1–11, 11 2019.

[Kim et al., 2019b] Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum (Proc. Eurographics 2019)*, 38(2):59–70, 5 2019.

[Kim et al., 2020] Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. Lagrangian Neural Style Transfer for Fluids. *ACM Transactions on Graphics (Proc. SIGGRAPH 2020)*, 39(4), 7 2020.

[Kim, 2008] Theodore Kim. Hardware-aware analysis and optimization of stable fluids. In *Proceedings of the Symposium on Interactive 3D Graphics and Games, I3D 2008*, pages 99–106, 2008.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, volume 5, 2015.

[Koschier and Bender, 2017] Dan Koschier and Jan Bender. Density maps for improved SPH boundary handling. In *Proceedings - SCA 2017: ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 1–10, 2017.

[Koschier et al., 2019] Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. Eurographics Tutorial 2019 — Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids. In *Eurographics 2019 - Tutorials*, 2019.

[Kulkarni et al., 2015] Tejas D. Kulkarni, William F. Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network. *Advances in Neural Information Processing Systems*, 2015-Janua:2539–2547, 3 2015.

[Kwatra et al., 2005] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. In *ACM SIGGRAPH 2005 Papers on - SIGGRAPH '05*, page 795, New York, New York, USA, 2005. ACM Press.

[Kwatra et al., 2006] Vivek Kwatra, David Adalsteinsson, Nipun Kwatra, Mark Carlson, and Ming C. Lin. Texturing fluids. In *ACM SIGGRAPH 2006: Sketches, SIGGRAPH '06*, page 63, New York, New York, USA, 2006. ACM Press.

[Ladický et al., 2015] L'ubor Ladický, Sohyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven Fluid Simulations using Regression Forests. *ACM Transactions on Graphics*, 34(6):1–9, 10 2015.

[Ledig et al., 2017] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings - 30th IEEE*

Conference on Computer Vision and Pattern Recognition, CVPR 2017, volume 2017-Janua, pages 105–114. IEEE, 7 2017.

[Li et al., 2018a] Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. Robust flow-guided neural prediction for sketch-based freeform surface modeling. *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, 37(6):238:1–238:12, 2018.

[Li et al., 2018b] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs. Data Reduction Techniques for Simulation, Visualization and Data Analysis. *Computer Graphics Forum*, 37(6):422–447, 2018.

[Li et al., 2018c] Tzu Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte Carlo ray tracing through edge sampling. In *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, pages 1–11, New York, New York, USA, 2018. ACM Press.

[Li et al., 2018d] Yanchun Li, Nanfeng Xiao, and Wanli Ouyang. Improved boundary equilibrium generative adversarial networks. *IEEE Access*, 6:11342–11348, 2018.

[Libersky and Petschek, 2008] Larry D. Libersky and A. G. Petschek. Smooth particle hydrodynamics with strength of materials. In *Advances in the Free-Lagrange Method Including Contributions on Adaptive Gridding and the Smooth Particle Hydrodynamics Method*, pages 248–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Lindstrom and Isenburg, 2006] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[Liu and Jacobson, 2019] Hsueh Ti Derek Liu and Alec Jacobson. Cubic stylization. *ACM Transactions on Graphics*, 38(6), 10 2019.

[Liu et al., 2015] Beibei Liu, Gemma Mason, Julian Hodgson, Yiying Tong, and Mathieu Desbrun. Model-reduced variational fluid simulation. *ACM Transactions on Graphics*, 34(6):244, 2015.

[Liu et al., 2018a] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image Inpainting for Irregular Holes Using Partial Convolutions. In *Eurographics Workshop on Computer Animation and Simulation 2018*, volume 11215 LNCS, pages 89–105. 2018.

[Liu et al., 2018b] Hsueh Ti Derek Liu, Michael Tao, and Alec Jacobson. Paparazzi: Surface editing by way of multi-view image processing. *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, 2018.

[Lombardi et al., 2019] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics*, 38(4):1–14, 7 2019.

[Long and Reinhard, 2009] Benjamin Long and Erik Reinhard. Real-time fluid simulation using discrete sine/cosine transforms. In *Proceedings of I3D 2009: The 2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 99–106, New York, New York, USA, 2009. ACM Press.

[Loper and Black, 2014] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision 2014*, volume 8695 LNCS, pages 154–169. 2014.

[Losasso et al., 2008] Frank Losasso, Jerry O. Talton, Nipun Kwatra, and Ron Fedkiw. Two-way coupled SPH and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):797–804, 7 2008.

[Loshchilov and Hutter, 2017] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 8 2017.

[Lu et al., 2016] Wenlong Lu, Ning Jin, and Ronald Fedkiw. Two-way Coupling of Fluids to Reduced Deformable Bodies. In *SCA '16: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 67–76, 2016.

[Lucy, 1977] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013, 12 1977.

[Lumley, 1967] John L. Lumley. The Strucure of Inhomogeneous Turbulent Flows. In A M Yaglom and V I Tatarski, editors, *Atmospheric Turbulence and Radio Wave Propagation*, pages 166–178. Nauka, 1967.

[Lun et al., 2018] Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks. In *Proceedings - 2017 International Conference on 3D Vision, 3DV 2017*, pages 67–77. IEEE, 2018.

[Ma et al., 2009] Chongyang Ma, Baining Guo, Li Yi Wei, and Kun Zhou. Motion Field Texture Synthesis. In *ACM Transactions on Graphics*, volume 28, pages 1–8. ACM, 2009.

[Ma et al., 2018] Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics*, 37(4), 2018.

[Maas et al., 2013] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, volume 30, page 3, 2013.

[Macklin and Müller, 2013] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics*, 32(4):1–104, 2013.

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

[Masci et al., 2015] Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. Geodesic Convolutional Neural Networks on Riemannian Manifolds. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015-Febru, pages 832–840, 2015.

[Mathieu et al., 2016] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[McGuire and Fein, 2006] Morgan McGuire and Andi Fein. Real-time rendering of cartoon smoke and clouds. In *NPAR Symposium on Non-Photorealistic Animation and Rendering*, volume 2006 of *NPAR '06*, pages 21–26. Association for Computing Machinery, 2006.

[McNamara et al., 2004] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *ACM SIGGRAPH 2004 Papers, SIGGRAPH 2004*, pages 449–456, New York, New York, USA, 2004. ACM Press.

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

[Monaghan, 1992] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 9 1992.

[Monaghan, 1994] J. J. Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110(2):399–406, 2 1994.

[Monaghan, 2005] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703–1759, 8 2005.

[Mordvintsev et al., 2018] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable Image Parameterizations. *Distill*, 3(7), 2018.

[Mordvintsev, 2016] Alexander Mordvintsev. DeepDreaming with TensorFlow, 2016.

[Morton et al., 2018] Jeremy Morton, Freddie D. Witherden, Mykel J. Kochenderfer, and Antony Jameson. Deep dynamical modeling and control of unsteady fluid flows. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 9258–9268. Curran Associates, Inc., 2018.

[Mullen et al., 2009] Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics*, 28(3):1, 2009.

[Müller et al., 2003] 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 2003*, 2003.

[Narain et al., 2007] Rahul Narain, Vivek Kwatra, Huai-Ping Lee, Theodore Kim, Mark Carlson, and Ming C. Lin. Feature-Guided Dynamic Texture Synthesis on Continuous Flows. In *Proc. Eurographics Symposium on Rendering (EGSR)*, EGSR'07, pages 361–370, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[Narain et al., 2008] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Transactions on Graphics*, 27(5):1, 12 2008.

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

[Nielsen et al., 2009] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Computer Animation, Conference Proceedings*, pages 217–226, New York, New York, USA, 2009. ACM Press.

[Nielsen et al., 2011] Michael B. Nielsen, Weta Digital, and Robert Bridson. Guide shapes for high resolution naturalistic liquid simulation. In *ACM Transactions on Graphics*, volume 30, page 1, New York, New York, USA, 2011. ACM Press.

[Nimier-David et al., 2019] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics*, 38(6):1–17, 11 2019.

[Nishida et al., 2016] Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Transactions on Graphics*, 35(4), 2016.

[Odena et al., 2017] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and Checkerboard Artifacts. *Distill*, 1(10):e3, 2017.

[Okabe et al., 2015] Makoto Okabe, Yoshinori Dobashi, Ken Anjyo, and Rikio Onai. Fluid volume modeling from sparse multi-view images by appearance transfer. *ACM Transactions on Graphics*, 34(4):93, 2015.

[Olah et al., 2017] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2(11), 2017.

[Osher and Sethian, 1988] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 11 1988.

[Pan and Manocha, 2016] Zherong Pan and Dinesh Manocha. Efficient Optimal Control of Smoke using Spacetime Multigrid. 8 2016.

[Pan and Manocha, 2017] Zherong Pan and Dinesh Manocha. Editing smoke animation using a deforming grid. *Computational Visual Media*, 3(4):369–378, 2017.

[Pan et al., 2013] Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. Interactive localized liquid motion editing. *ACM Transactions on Graphics*, 32(6):1–10, 11 2013.

[Paszke et al., 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[Peer et al., 2015] Andreas Peer, Markus Ihmsen, Jens Cornelis, and Matthias Teschner. An implicit viscosity formulation for SPH fluids. *ACM Transactions on Graphics*, 34(4):1–10, 2015.

[Peng et al., 2018] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van De Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4):1–14, 8 2018.

[Perlin, 1985] Ken Perlin. Image Synthesizer. *Computer Graphics (ACM)*, 19(3):287–296, 7 1985.

[Pfaff et al., 2009] Tobias Pfaff, Nils Thuerey, Markus Gross, Andrew Selle, and Markus Gross. Synthetic Turbulence using Artificial Boundary Layers. *ACM Transactions on Graphics*, 28(5):1–10, 12 2009.

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

[Prantl et al., 2019] Lukas Prantl, Boris Bonev, and Nils Thuerey. Generating liquid simulations with deformation-aware neural networks. *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[Qi et al., 2016] Charles R. Qi, Hao Su, Matthias Niebner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view CNNs for object classification on 3D data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:5648–5656, 4 2016.

[Qi et al., 2017] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 77–85, 2017.

[Rasmussen et al., 2004] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Computer Animation 2004 - ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 193–202, New York, New York, USA, 2004. ACM Press.

[Raveendran et al., 2012] Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. Controlling liquids using meshes. In *Computer Animation 2012 - ACM SIGGRAPH / Eurographics Symposium Proceedings, SCA 2012*, SCA '12, pages 255–264, Goslar Germany, Germany, 2012. Eurographics Association.

[Raveendran et al., 2014] Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. Blending liquids. *ACM Transactions on Graphics*, 33(4):137, 2014.

[Reinhardt et al., 2019] Stefan Reinhardt, Tim Krake, Bernhard Eberhardt, and Daniel Weiskopf. Consistent shepard interpolation for SPH-based fluid animation. *ACM Transactions on Graphics*, 38(6), 2019.

[Ren et al., 2013] Bo Ren, Chen Feng Li, Ming C. Lin, Theodore Kim, and Shi Min Hu. Flow field modulation. *IEEE Transactions on Visualization and Computer Graphics*, 19(10):1708–1719, 2013.

[Ren et al., 2014] Bo Ren, Chenfeng Li, Xiao Yan, Ming C. Lin, Javier Bonet, and Shin Min Hu. Multiple-fluid SPH simulation using a mixture model. *ACM Transactions on Graphics*, 33(5):1–11, 9 2014.

[Ruder et al., 2018] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic Style Transfer for Videos and Spherical Images. *International Journal of Computer Vision*, 126(11):1199–1219, 11 2018.

[Sato et al., 2018a] Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. Example-based turbulence style transfer. *ACM Transactions on Graphics*, 37(4):84, 2018.

[Sato et al., 2018b] Syuhei Sato, Yoshinori Dobashi, and Tomoyuki Nishita. Editing fluid animation using flow interpolation. *ACM Transactions on Graphics*, 37(5):1–12, 2018.

[Schechter and Bridson, 2008] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Computer Animation 2008 - ACM SIGGRAPH / Eurographics Symposium, SCA 2008 - Proceedings*, pages 1–7, 2008.

[Schenck and Fox, 2018] Connor Schenck and Dieter Fox. SPNets: Differentiable Fluid Dynamics for Deep Neural Networks. 6 2018.

[Schmitzer, 2019] Bernhard Schmitzer. Stabilized sparse scaling algorithms for entropy regularized transport problems. *SIAM Journal on Scientific Computing*, 41(3):A1443–A1481, 2019.

[Selle et al., 2004] Andrew Selle, Alex Mohr, and Stephen Chenney. Cartoon rendering of smoke animations. In *NPAR Symposium on Non-Photorealistic Animation and Rendering*, NPAR '04, pages 57–60. Association for Computing Machinery, 2004.

[Selle et al., 2005] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics*, 24(3):910–914, 7 2005.

[Selle et al., 2008] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable MacCormack method. *Journal of Scientific Computing*, 35(2-3):350–371, 6 2008.

[Shen et al., 2020] Yuefan Shen, Changgeng Zhang, Hongbo Fu, Kun Zhou, and Youyi Zheng. DeepSketchHair: Deep Sketch-based 3D Hair Modeling. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2020.

[Shi and Yu, 2005a] Lin Shi and Yizhou Yu. Controllable smoke animation with guiding objects. *ACM Transactions on Graphics*, 24(1):140–164, 2005.

[Shi and Yu, 2005b] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *Computer Animation, Conference Proceedings*, pages 229–236, New York, New York, USA, 2005. ACM Press.

[Simo-Serra et al., 2016] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. Learning to simplify: Fully convolutional networks for rough sketch cleanup. *ACM Transactions on Graphics*, 35(4):1–11, 2016.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 9 2015.

[Smith, 2017] Leslie N. Smith. Cyclical learning rates for training neural networks. In *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pages 464–472, 2017.

[Solenthaler and Pajarola, 2009] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible SPH. In *ACM Transactions on Graphics*, volume 28, pages 1–40, 2009.

[Solomon et al., 2015] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics*, 34(4), 2015.

[spi, 2015] Spiral Image Source (http://ardezart.com/?attachment_id=252), 2015.

[Stam, 1999] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999*, pages 121–128, New York, New York, USA, 1999. ACM Press.

[Stanton et al., 2013] Matt Stanton, Yu Sheng, Martin Wicke, Federico Perazzi, Amos Yuen, Srinivasa Narasimhan, and Adrien Treuille. Non-polynomial galerkin projection on deforming meshes. *ACM Transactions on Graphics*, 32(4):86:1, 2013.

[Stanton, 2014] Matthew Luchak Stanton. *Data-Driven Methods for Interactive Simulation of Complex Phenomena*. PhD thesis, Carnegie Mellon University, 2014.

[Steinhoff and Underhill, 1994] John Steinhoff and David Underhill. Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 6(8):2738–2744, 8 1994.

[sti, 2018] Style Image Source (https://github.com/titu1994/Neural-Style-Transfer), 2018.

[Stokes, 1845] George Gabriel Stokes. On the Theories of the Internal Friction of Fluids in Motion, and of the Equilibrium and Motion of Elastic Solids. In *Mathematical and Physical Papers vol.1*, pages 75–129. Cambridge University Press, Cambridge, 1845.

[Stomakhin et al., 2013] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics*, 32(4), 2013.

[Suwajanakorn et al., 2017] Supasorn Suwajanakorn, Steven M. Seitz, and Ira Kemelmacher-Shlizerman. Synthesizing obama: Learning lip sync from audio. *ACM Transactions on Graphics*, 36(4):1–13, 7 2017.

[Szegedy et al., 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9, 2015.

[Talmi et al., 2017] Itamar Talmi, Roey Mechrez, and Lihi Zelnik-Manor. Template matching with deformable diversity similarity. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 1311–1319, 2017.

[Témam, 1969] R. Témam. Sur l'approximation de la solution des équations de Navier-Stokes par la méthode des pas fractionnaires (I). *Archive for Rational Mechanics and Analysis*, 32(2):135–153, 1 1969.

[Thuerey and Pfaff, 2018] Nils Thuerey and Tobias Pfaff. MantaFlow, 2018.

[Thuerey, 2016] Nils Thuerey. Interpolations of smoke and liquid simulations. *ACM Transactions on Graphics*, 36(1):3, 2016.

[Thürey et al., 2006] N Thürey, R Keiser, M Pauly, and U Rüde. Detail-Preserving Fluid Control. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, page 7–12. Eurographics Association, 2006.

[Tompson et al., 2019] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, volume 70, pages 3424–3433, 2019.

[Treuille et al., 2003] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. *ACM Transactions on Graphics*, 22(3):716–723, 7 2003.

[Treuille et al., 2006] Adrien Treuille, Andrew Lewis, and Zoran Popović. Model reduction for real-time fluids. *ACM Transactions on Graphics*, 25(3):826–834, 2006.

[Tulsiani et al., 2017] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:209–217, 4 2017.

[Ulyanov et al., 2016] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. 7 2016.

[Um et al., 2014] Kiwon Um, Seungho Baek, and Junghyun Han. Advanced Hybrid Particle-Grid Method with Sub-Grid Particle Correction. *Computer Graphics Forum*, 33(7):209–218, 10 2014.

[Um et al., 2018] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid Splash Modeling with Neural Networks. In *Computer Graphics Forum*, volume 37, pages 171–182. Wiley Online Library, 2018.

[Umetani and Bickel, 2018] Nobuyuki Umetani and Bernd Bickel. Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics*, 37(4):1–10, 7 2018.

[Velinov et al., 2018] Zdravko Velinov, Marios Papas, Derek Bradley, Paulo Gotardo, Parsa Mirdehghan, Steve Marschner, Jan Novák, and Thabo Beeler. Appearance capture and modeling of human teeth. In *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*, pages 1–13, New York, New York, USA, 2018. ACM Press.

[Vincent et al., 2010] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre Antoine Manzagol. Stacked denoising autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010.

[Wang et al., 2019] Tuanfeng Y. Wang, Duygu Ceylan, Jovan Popović, and Niloy J. Mitra. Learning a shared shape space for multimodal garment design. *ACM Transactions on Graphics*, 37(6):1–13, 2019.

[Weißmann and Pinkall, 2010] Steffen Weißmann and Ulrich Pinkall. Filament-based smoke with vortex shedding and variational reconnection. *ACM SIGGRAPH 2010 Papers, SIGGRAPH 2010*, 29(4):1, 2010.

[Wicke et al., 2009] Martin Wicke, Matt Stanton, and Adrien Treuille. Modular bases for fluid dynamics. In *ACM SIGGRAPH 2009 papers on - SIGGRAPH '09*, page 1, New York, New York, USA, 2009. ACM Press.

[Wiewel et al., 2019] S. Wiewel, M. Becher, and N. Thuerey. Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. *Computer Graphics Forum*, 38(2):71–82, 2 2019.

[Wu et al., 2015] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1912–1920, 2015.

[Xie et al., 2018] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. TempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Transactions on Graphics*, 37(4), 2018.

[Yan et al., 2016] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3D supervision. *Advances in Neural Information Processing Systems*, pages 1704–1712, 12 2016.

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

[Yang et al., 2016] Cheng Yang, Xubo Yang, and Xiangyun Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

[Yifan et al., 2019] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics*, 38(6), 6 2019.

[Yu et al., 2011] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Lagrangian texture advection: Preserving both spectrum and velocity field. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1612–1623, 11 2011.

[Yuan et al., 2011] Zhi Yuan, Fan Chen, and Ye Zhao. Pattern-Guided Smoke Animation with Lagrangian Coherent Structure. *ACM Transactions on Graphics*, 30(6):1–8, 2011.

[Zhao et al., 2016] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2016.

[Zhu and Bridson, 2005] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Transactions on Graphics*, 24(3):965–972, 7 2005.

[Zhu et al., 2011] Bo Zhu, Michiaki Iwata, Ryo Haraguchi, Takeo Igarashi, and Nobuyuki Umetani. Sketch-based Dynamic Illustration of Fluid Systems. *ACM Transactions on Graphics*, 30(6):1–8, 2011.