Editing Fluid Simulations with Jet Particles

Julian Hodgson

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Engineering

at

University College London.

Department of Computer Science

University College London

March 5, 2018

I, Julian Hodgson, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Fluid simulation is an important topic in computer graphics in the pursuit of adding realism to films, video games and virtual environments. The results of a fluid simulation are hard to edit in a way that provide a physically plausible solution. Edits need to preserve the incompressibility condition in order to create natural looking water and smoke simulations.

In this thesis we present an approach that allows a simple artist-friendly interface for designing and editing complex fluid-like flows that are guaranteed to be incompressible in two and three dimensions. Key to our method is a formulation for the design of flows using *jet particles*.

Jet particles are Lagrangian solutions to a regularised form of Euler's equations, and their velocity fields are divergence-free which motivates their use in computer graphics. We constrain their dynamics to design divergence-free flows and utilise them effectively in a modern visual effects pipeline. Using just a handful of jet particles we produce visually convincing flows that implicitly satisfy the incompressibility condition. We demonstrate an interactive tool in two dimensions for designing a range of divergence-free deformations.

Further we describe methods to couple these flows with existing simulations in order to give the artist creative control beyond the initial outcome. We present examples of local temporal edits to smoke simulations in 2D and 3D. The resulting methods provide promising new ways to design and edit fluid-like deformations and to create general deformations in 3D modelling.

We show how to represent existing divergence-free velocity fields using jet particles, and design new vector fields for use in fluid control applications.

Finally we provide an efficient implementation for deforming grids, meshes, volumes, level sets, vectors and tensors, given a jet particle flow.

Acknowledgements

I would like to acknowledge and thank my supervisor Anthony Steed for steady and solid support and advice, and my second supervisor Niloy Mitra for suggesting the topic of fluid editing and for many motivating discussions during the project. I would also like to thank Nils Thuerey for his support providing expert advice in fluid dynamics and for his enthusiastic feedback.

I am indebted to Daryl Holm for inviting me to take his Geometric Mechanics course at Imperial College and for my introduction to his group. Here I met Henry Jacobs who I would like to thank for an explanation of jet particles, providing python code for evaluating the jet particle flow, and many helpful discussions on the subject. Also from this group I would like to thank Dmitri Pavlov for discussions on structure preserving flow, and Jaap Eldering for discussions on geometric mechanics and providing useful feedback. Thanks also to Stefan Sommer for some valuable conversations about jet particles.

From my time at Cinesite I would like to thank Jonathan Davies for his Houdini expertise and valuable production experience, Don Boogert for many helpful technical discussions and support on the Cinesite pipeline, and my industrial supervisor Michele Sciolette.

I would like to thank Andrew Ruhemann for sponsoring me at Passion Pictures during the first part of my EngD, and Hugo Sands for his great efforts in helping make the EngD work at Passion.

I would like to recognise the fantastic support of the 1851 Royal Commission for sponsorship with their Industrial Fellowship, and the EPSRC for funding the EngD at UCL.

I would like also to thank Cristin Barghiel at SideFX Software for providing me with a research license for Houdini FX for simulation and rendering, and Alan Trombla for donating an RV license for image and movie conversion and editing.

And I would like in particular to thank my wife Giovanna and my family who have supported me throughout this adventure.

Contents

1 Introduction **17**

1.1 Motivation 17

1.2 Jet Particles 18

1.3 Hypotheses 19

1.4 Chapter Overview 20

2 Background **22**

2.1 Grid Based Solvers 22

2.2 Smooth Particle Hydrodynamics 23

2.3 Particle in Cell Methods and FLIP 24

2.4 Fluid Control 25

2.5 Model Reduction 26

2.6 Turbulence 29

2.7 Fluid Editing 30

2.8 Space-time Methods 32

2.9 LDDMM 33

2.10 Jet Particles 34

2.11 Vector Field Design 35

3 Technical Background **37**

3.1 Nomenclature 37

3.2 Basic Definitions 38

3.3 Introduction 39

3.4 Matrix-valued Kernels 40
3.5 Hamiltonian Dynamics 43
3.6 Jet Particles 46
3.6.1 Basic Definitions 46
3.6.2 Equations of Motion 47
3.6.3 Simulation Example 48
3.7 Relationship with Eulerian Flow 50

4 Sketching Flows with Jet Particles **52**
4.1 Sketching Flows Interactively 52
4.2 Deformation with Constrained Particles 53
4.3 Constrained 0-Jet Particles 54
4.4 Fluid Sketching Application 55
4.5 Constrained 1-Jet Particles 57
4.6 Keyframe Generation 60
4.7 Dynamics of Jet Particles 63
4.8 Conclusions and Future Work 66

5 Editing Fluids with Jet Particles **69**
5.1 Editing Existing Flows 70
5.1.1 Comparison with Skinning 71
5.1.2 Deformation Metric 73
5.1.3 Deforming 2D Smoke Simulations 73
5.1.4 Deforming 3D Smoke Simulations 77
5.2 Tracking Jet Particles 80
5.3 Advecting Jet Particles 83
5.4 Push-forward of Jet Particles 84
5.5 Deformation Keyframes 86
5.6 Jet Particle Rigs in Maya 92
5.7 Conclusions and Future Work 95
5.7.1 Coupling Rigid Bodies with Fluids 95

5.7.2 Up-Resing Existing Flows 95

6 Designing Vector Fields 97

6.1 Projection 97

6.1.1 Motivation 98

6.1.2 Project Velocity Field onto 0-Jet Particles 99

6.1.3 Project Velocity Field onto 1-Jet Particles 100

6.2 Vector Field Interpolation 101

6.2.1 Examples 102

6.2.2 Results 103

6.2.3 Conclusions 104

6.2.4 Future Work 109

6.3 Vector Field Design 109

6.3.1 Tangent Constraints 110

6.3.2 Deformation Gradient Constraints 112

6.3.3 Controlling the Velocity and Gradient 113

6.3.4 Varying the Kernel Size 115

6.4 Conclusions and Future Work 117

6.4.1 Representing Existing Velocity Fields with Jet Particles . . . 117

6.4.2 Designing Flows on Surfaces 118

6.4.3 Jet Particle Flow Rigs 118

7 Deformations with Jet Particles 120

7.1 Forwards Deformation 120

7.2 Backwards Flow 124

7.3 Coarse Grid Optimisation 126

7.4 Bezier Interpolation 128

7.5 Point Cloud Deformation 133

7.6 Conclusions 137

8 Conclusions 139

8.1 Summary 139

8.2 Future Work . . . 141
8.2.1 Deformation . . . 141
8.2.2 Jet Particle Simulations . . . 141

Appendices **143**

A Tools **144**
A.1 Manta . . . 144
A.2 Tensor Library . . . 144
A.3 OpenVDB . . . 145
A.3.1 Deformation of OpenVDB Files . . . 146
A.3.2 Visualisation in Maya . . . 151
A.3.3 Deformation in Maya Rig . . . 151
A.3.4 Calculating the Projection Matrix . . . 152
A.3.5 Houdini Implementation . . . 154

Bibliography **154**

List of Figures

3.1 The vector fields of the matrix kernel from Equation (3.2) for $a = 0, 0.5, 1.0$ with momenta $(1,0)$ inserted, showing the Gaussian, interpolated and divergence-free kernels from left to right 42

3.2 Hamiltonian particles. The arrows represent the momenta of the particles. 45

3.3 A single 1-jet particle in 2D with momentum $p = 0$, and μ taking the values *spin*, *stretch* and *shear*, illustrated from left to right 49

3.4 The velocity field generated by several 1-Jet particles. The colours of the particles represent the vorticity of the particles, with red and blue being positive and negative vorticity. 50

4.1 A constrained particle with a scalar kernel sketching a flow deforming a grid 56

4.2 A constrained 0-jet particle sketching a flow deforming a grid 56

4.3 A constrained 0-jet particle sketching a flow deforming a grid with $a = 0.5$ 56

4.4 Streamlines of 0-jet particles in 3D with momenta along the x, y and z axes, from left to right. 57

4.5 A constrained 1-jet *Spin* particle sketching a flow deforming a grid . 59

4.6 A constrained 1-jet *Scale* particle sketching a flow deforming a grid 59

4.7 A constrained 1-jet *Shear* particle sketching a flow deforming a grid 59

4.8 1-Jet Particles in 3D. Top row Scale S_{xy}, S_{xz}, S_{yz}, middle row Rotation R_x, R_y, R_z and bottom row Shear Sh_1, Sh_2, Sh_3. 61

4.9 Smoke density and velocity field edited with 0-jet and 1-jet particle . 62

4.10 0-Jet particle instabilities. A simulation of several 0-jet particles displaying instabilities. 64
4.11 1-Jet particle simulation. A simulation of several 1-jet particles displaying instabilities. 64
4.12 Initial conditions for the 0-jet particles 65
4.13 Unconstrained (left) vs constrained (right) 0-jet simulation after identical initial conditions. The constraint targets the particle momenta towards $(1,0)$. 65
4.14 The velocity field from a simulation of 30 0-jet particles interacting in 3D. 67

5.1 Comparison of a 0-jet flow with skinning by translation. The skinning deformation is on the left and the 0-jet particle flow is on the right. 72
5.2 Comparison of a 1-jet flow with skinning by rotation. The skinning deformation is on the left and the 1-jet particle flow is on the right. . 72
5.3 Editing of the 2D smoke simulation with a 0-jet particle flow 75
5.4 Editing of the 2D smoke simulation with a 1-jet particle flow 76
5.5 Deformation of a 3D smoke simulation with a 0-jet particle flow . . 78
5.6 Deformation of a 3D smoke simulation with 1-jet particle flow . . . 79
5.7 The new jet particles $L_\alpha(s)$ are created by offsetting the jet particles $J_\alpha(s)$ with the displacement vector $dq_\alpha(t)$, given by the flow φ_t applied at $q_\alpha(0)$ 81
5.8 Deformation of a 3D smoke simulation using advected jet particle flow, showing 8 selected frames in increasing time in left-right minor and top-bottom major order. 83
5.9 The new jet particle flow $L_\alpha(s)$ is calculated by deforming the jet particle flow $J_\alpha(s)$ with the cached flow φ_t 86
5.10 Deformation of an existing flow with turbulent 1-jet particles. The original simulation density at a frame is shown on the right, with the resulting deformed density displayed on the left. 87

5.11 The spline function β used for retracting the deformation. 88
5.12 Retracting the edit flow $\psi(s)$ around time T, by applying the curve β to the edit time s in the jet particle flow $L_\alpha(s,t)$. 88
5.13 A 2D smoke simulation deformed by a retracted 0-jet edit flow. The advected jet particle flow path before retraction is indicated by dots. Frames are in left-right minor and top-bottom major order. 90
5.14 A 2D smoke simulation deformed by a retracted 1-jet edit flow. The advected jet particle flow path is indicated with a dot. Frames are in left-right minor and top-bottom major order. 91
5.15 Smoke densities from a simulated smoke stack used in a shot in production. 93
5.16 The deformed smoke densities found by applying the jet particle flow. 93
5.17 The jet particle rig in Maya, used to deform the smoke densities in the simulation. The jet particle is constrained to a curve which is placed and edited interactively. A deformed version of the densities is shown in the viewport. 94

6.1 The known test velocity fields 103
6.2 0-Jet particles representing a constant field with $\sigma \in [0.25, 5]$ 105
6.3 0-Jet particles representing a divergence-free rotational field with $\sigma \in [0.25, 10]$ 106
6.4 1-Jet particles representing a constant field with $\sigma \in [0.25, 4]$ 107
6.5 1-Jet particles representing a divergence-free rotational field with $\sigma \in [0.25, 5]$ 108
6.6 Vector field matching curve tangents, $\sigma = 0.8$ 110
6.7 Vector field matching curve tangents, $\sigma = 0.2$ 111
6.8 Vector field matching velocities along a closed curve 111
6.9 Vector field given by aligning $\mu = R_y$ with the Frenet frame 113
6.10 Vector field given by aligning μ with the Frenet frame and matching u on γ 115
6.11 Vector field generated by 0-jet particles with varying σ_β 116

6.12 Vector field generating a tornado effect using eight 1-jet particles with varying σ_β 117

7.1 Deformation of rest grid with a 0-jet particle 122
7.2 The Jacobian matrix maps the rest triangle T_R to the triangle T deformed by the flow φ 123
7.3 Flow by φ mapping x' to x 125
7.4 Bilinear interpolation to estimate undeformed position q' from q using frames Q_{ij} and positions X_{ij} 127
7.5 Control points for a single bezier patch in 2D given the reverse flow $\varphi(\cdot,t)^{-1}$ 129
7.6 B-Spline surface consisting of four surface patches B_{ij} constructed with frames Q_{ij} 130
7.7 Control points for a slice of a 3D bezier spline 134
7.8 Deforming a point p with a flow of a point cloud given by closest *deform particles* $\alpha_i := (X_i, Q_i)$ 136

A.1 The deformation algorithm preserves the sparsity of the density field 149
A.2 The delta vectors used for the deformation. These are defined sparsely on a coarser grid than the data set being deformed. 150

List of Tables

3.1 An explanation of the symbols used throughout this document . . . 37

4.1 A basis for the Lie Algebra $\mathfrak{sl}_3(\mathbb{R})$. Note we mark S_{yz} as red since we must omit one element to form a basis. 60

List of Videos

4.1 Sketching with a scalar kernel 55
4.2 Sketching with a divergence-free matrix kernel 55
4.3 Sketching with an interpolated matrix kernel 55
4.4 2D smoke editing application 61
4.5 0-Jet particle simulation 63
4.6 1-Jet particle simulation 63
4.7 Jet particle constraints 65

5.1 Comparison of 0-Jet flow with skinning by translation 71
5.2 Comparison of 1-jet flow with skinning by rotation 71
5.3 Original 2D Smoke Simulation 74
5.4 Deformation of 2D Smoke with Static 0-jet particle 74
5.5 Original 3D smoke simulation 77
5.6 3D smoke deformed with 0-jet particle 77
5.7 3D smoke deformed with 1-jet particle 77
5.8 3D smoke deformed with an advected 0-jet particle 81
5.9 3D smoke deformed with an advected 1-jet particle 82
5.10 3D smoke deformed with multiple advected 1-jet particles 82
5.11 Deforming an existing smoke simulation with a coupled 1-jet particle flow 86
5.12 2D smoke 0-jet deformation keyframe edit 89
5.13 2D smoke 1-jet deformation keyframe edit 89

Chapter 1

Introduction

1.1 Motivation

Since the introduction of grid-based fluid solvers to the graphics community by Foster and Metaxas 1996 and Stam 1999, fluid simulations have been used extensively to create content for virtual worlds. Mathematical models generate physically plausible results and can add a level of realism that would otherwise be hard to achieve.

However, fluid simulations are hard to predict and control, and are expensive to compute.

If an artist needs to make a small change to some aspect of a fluid simulation, this typically requires the adjustment of constraints and the re-computation of the equations of motion, which can cause substantial delays. In this thesis we propose techniques that allow the artist to edit existing fluid simulations by enabling the user to design and incorporate new volume preserving flows.

The principal contributions of this thesis are:

1. Finding a natural and computationally efficient way to generate volume preserving flows that "behave like a fluid"

2. Coupling the new flow with an existing off-line simulation for fluid editing

3. Creating control rigs for designing and editing divergence-free velocity fields for fluid control

4. Implementation of deformations given by jet particle flow on sets of points, vectors, volumes, meshes and frames

Our solution to (1) is obtained by using Lagrangian methods based upon machinery from the world of diffeomorphic medical image registration, in order to create time-dependent deformations which are "fluid-like". By using *jet particles* we are able to give the user direct control over the deformation gradient at the particle positions.

We provide a solution to editing existing fluid flows (2) by advecting jet particles and their momenta with a divergence-free velocity field. The source of this velocity field can be an existing off-line flow or a live fluid simulation.

In order to address (3) we propose methods for designing static divergence-free velocity fields using jet particles that satisfy constraints on their gradients at the particle positions. We outline how this may be useful for fluid control and compression of existing divergence-free velocity fields.

Finally we identify as contribution (4) the development of efficient solutions for implementing the above deformations given by the jet particle flow. These methods take advantage of the Hamiltonian nature of the flow to create large deformations without loss in accuracy.

As a secondary aim, this thesis also brings jet particle dynamics to the graphics community.

1.2 Jet Particles

In mathematics, the k-jet of a function is a polynomial that matches the function up to order k, found by truncating its Taylor series. On the other hand, jet particles are Hamiltonian particles that carry higher order information about the flow with them. Each particle carries information describing the local deformation of the flow up to order k, and so they are named k-jet particles.

The dynamics of these particles serves as a model for generating regularised incompressible fluid flow. The flows generated by these particles can be evaluated without a velocity projection step, and the particles generate divergence-free veloc-

ity fields, hence they create incompressible flows by definition.

Jet particles are a promising area to explore in computer graphics, and we hope to demonstrate this by identifying and implementing natural and beneficial examples of their use in fluid sketching and editing.

Although the main application of this thesis is to design and edit incompressible fluid flow, the deformation techniques we describe are also well suited for shape deformation, and may have a substantial role to play in other areas of computer graphics.

We outline our first three contributions as Hypotheses 1,2 and 3 in the following section.

1.3 Hypotheses

Hypothesis 1. Sketching Flows

We assert that jet particles can be used in a natural and efficient way to sketch realistic flows. By constraining jet particles to move with an artist's brush stroke, new fluid-like flows can be designed easily and intuitively.

We demonstrate our system in Chapter 4 by creating a real-time application for sketching fluid-like deformations. Further, we describe how to constrain simulations of multiple jet particles in order to create more complex flows.

Hypothesis 2. Editing Simulations

We assert that jet particles bring a natural way to edit and augment existing flows. By creating sketch-based edit flows at key frames in an off-line fluid simulation, new physically plausible fluid states can be created using jet particle flow. Natural variations in the original fluid simulation can be produced by composing the original flow with the sketched edit flow.

In Chapter 5 we propose different methods for coupling our designer flows given by jet particles with cached simulated flows. Firstly we show how to deform the flow by deforming the simulation space directly. Then we show how to track the coordinate system of the sketched flow with the cached flow using advection. Finally we develop a deeper coupling method where we generate a family of

deformations linked with the original flow, including a method for retracting the deformation over time.

We show examples by means of smoke simulations including the use of scenes generated from real visual effects shots in production.

Hypothesis 3. Designing Divergence-free Vector Fields

We assert that jet particles provide an intuitive system for designing and editing divergence-free vector fields, that allows us direct control over the deformation gradient at key positions in the fluid. Configurations of static jet particles create an interpolated vector field over the surrounding space at arbitrary resolutions, and their momenta can be manipulated to edit the vector field. Further, existing divergence-free vector fields can be represented by jet particles.

In Chapter 6 we show how to project an existing divergence-free velocity field onto arrangements of jet particles. We show how to modify the vector field around the particles using the extra degrees of freedom provided by the 1-jet particles. In this sense we create a flow control rig using jet particles for use in animation and fluid control systems.

We note the difference between divergence-free vector fields and incompressible flows and refer the reader to Definition 3.5 to make clear the links between the two. Although controlling each has its own challenges, they are both valuable editing tools for use in fluid control and editing. We must design incompressible flows in order to edit an existing fluid simulation over time. But static divergence-free vector fields can be used as velocity constraints as well.

1.4 Chapter Overview

After we present the relevant background work in Chapter 2, this thesis begins with an introduction to matrix kernels and jet particle dynamics in Chapter 3. The Sections therein introduce the mathematics that we shall use in the following chapters, and are included here since we have not seen their use in computer graphics applications to date.

The substantive contributions from his thesis are presented in Chapters 4, 5, 6 and 7.

1. In Chapter 4 we introduce methods for designing smooth divergence-free flows by using curves sketched by the user, in order to constrain jet particle to create realistic flows.

2. In Chapter 5 we provide methods for editing fluid simulations with our sketched flows, by propagating the jet particle's deformation into existing off-line flows.

3. In Chapter 6 we show how to use jet particles to design vector fields for fluid control, by solving for configurations of jet particles that have prescribed velocities and velocity gradients at given particle positions.

4. In Chapter 7 we present algorithms for implementing the deformations that are required in the earlier chapters. This includes direct evaluation techniques, as well as techniques for approximating the deformation on coarser grids, point clouds and a higher order technique using splines.

We present the conclusions in Chapter 8 and we also suggest directions for future research that builds on the work in this thesis.

Finally in Appendix A, we give details of the tools we have developed and how they were used in practice in order to create our examples.

Chapter 2

Background

Fluid simulation was introduced to the graphics community by Foster and Metaxas [1] and further popularised with the stable semi-Lagrangian advection algorithm of Stam [2]. Complementary to these Eulerian, grid based approaches to fluid simulation are the Lagrangian particle based methods such as Smooth Particle Hydrodynamics [3] (see below). Both of these methods are effective in generating realistic solutions for the purposes of computer graphics simulation.

Additionally there are hybrid schemes such as the Particle In Cell method, which add a grid for use in the particles' velocity projection step, and FLIP methods that improve on this technique by reducing dissipation. See Zhu and Bridson [4] for their introduction to computer graphics and references therein.

2.1 Grid Based Solvers

Foster and Metaxas [1] introduced a grid based solver for simulating liquids based on the Navier-Stokes equations. The domain and boundaries are discretised into a regular grid, and an iterative finite difference method is required to find new velocities at each time step that satisfy the divergence-free condition.

Stam [2] improve the stability of the advection with a technique known as the *method of characteristics*, using a semi-Lagrangian scheme where the advected quantities are calculated by tracing back the properties of fluid parcels upstream using the velocity field. Additionally they solve the Poisson equation when calculating the velocity projection, improving the accuracy of this step. How-

ever these improvements come at the expense of increasing numerical dissipation. Lentine, Grétarsson, and Fedkiw [5] improve the accuracy and stability of the semi-Lagrangian method by combining the scheme with an additional forwards advection step, to guarantee the conservation of mass and momentum in incompressible flows.

Fluid flows on surfaces were introduced to the graphics community by Stam [6] who created a stable Navier-Stokes solver on subdivision surfaces. Still in the discrete surface setting, fluid flow is simulated on arbitrary simplicial complexes [7]. The action of flows on differential 1-forms can be calculated on regular grids [8]. This relates to our work as we seek to deform grids carrying density and vector information with the flows designed in this thesis.

Solvers are not required to store velocities directly on the grid. As an example, De Witt, Lessig, and Fiume [9] introduce a spectral method for solving the Navier-Stokes equations using a spectral basis for the velocity calculated using eigenfunctions of the Laplacian operator. In another direction, Azencot et al. demonstrate a method for modelling fluid flow on triangle meshes using a variational integrator storing vorticity, which conserves vorticity by construction.

2.2 Smooth Particle Hydrodynamics

Smooth Particle Hydrodynamics (SPH) is a Lagrangian approach to simulating fluids which conserves mass that is carried on a discrete set of particles. The particles carry additional physical values which are interpolated through the domain together with their gradients, using a scalar smoothing kernel. See Ihmsen et al. [11] for a recent STAR report.

SPH was initially developed in the astrophysics community by Monaghan [3]. It was first introduced into the animation community by Desbrun and Gascuel [12], who demonstrate the technique by generating the flow of highly deformable objects. The authors Müller, Charypar, and Gross [13] showed how the technique can be used to simulate water with a free surface, whilst Alduán and Otaduy [14] show how to apply SPH to the simulation of granular solids such as sand.

Using volume fractions to model multiphase flow allows for the mixing of

different fluids [15], and this approach is later extended to incorporate the coupling between liquids and solids [16].

Recent advances in SPH include the use of a novel unified particle type, introduced for representing fluids and liquid boundaries, and enabling phase change effects to be implemented easily in the solver [17]. Further, a recent step forwards in SPH simulation has brought speed improvements of up to two orders of magnitude by avoiding the simulation of ambient air particles [18]. Moreover an adaptive SPH solver utilising a continuous distribution of particle sizes has been proposed, allowing the efficient simulation of liquids using a range of resolutions [19].

Typically the SPH kernels should be close to Gaussian [3], although there are a number of other properties that are desirable in the kernel function. These properties are unity, compact support, positivity, decay, delta function property, symmetry and smoothness [20].

The pressure at each particle is calculated based on the local change in density from the original, and the gradient of this pressure drives the advection of the particles in a way that acts to restore the original density, and thus enforces incompressibility.

The velocities calculated in an SPH solver are not divergence-free without taking additional steps, and so the flow generated will be compressible as the fluid oscillates in a way to target its original density [11].

One issue with SPH is that its accuracy is proportional to the gradient in the velocity field, and this in turn is sensitive to the spacing of the particles. Further, SPH simulations can experience bunching of particles, so the accuracy is bound to drift during simulation.

2.3 Particle in Cell Methods and FLIP

Particle in Cell (PIC) methods involve seeding the fluid domain with particles, by adding around 8 to 12 particles placed randomly within each cell [21]. See Zhu and Bridson [4] for their introduction to computer graphics.

At each time step the particle velocities are transferred to a grid using linear

interpolation. The forces are integrated on this grid into velocities, and the velocity field is made divergence-free. Finally the velocities are then interpolated back onto the particles. This process of moving data back and forth between the particles and the grid each time step creates extra numerical viscosity which is mostly undesirable.

This decoupling of the particles from the grid is an example of a hybrid solver. The particles take on the task of the advection term, and the grid is responsible for the pressure projection.

The FLIP solver improves on PIC by mostly eliminating the loss of accuracy that occurs during the transfer of data between the particles and the grid. To achieve this, it is the difference between the original velocities that is interpolated back onto the particles and used in the velocity update. In this way the loss of accuracy occurs in the velocity deltas, and there is no smoothing of the original component of the velocity.

Zhu and Bridson [4] suggest calculating both the PIC and the FLIP velocity on the particles and allowing the user to interpolate between them in order to be able to control the amount of numerical viscosity for artistic effect.

Hybrid solvers also exist where the FLIP particles velocities are transferred to a particle grid consisting of SPH particles, and so the pressure calculation is performed on a Lagrangian grid [22]. For liquid simulations the number of particles required in the simulation can be significantly reduced if the particles are restricted to a narrow band around the surface [23].

2.4 Fluid Control

In this section we review existing methods for controlling fluids and see how jet particles can offer a novel approach.

Flow primitives such as sources, sinks and vortices are combined in two dimensions to design arbitrary divergence-free flows [24]. We can also use stationary jet particles as flow primitives in two and three dimensions to create divergence-free flows. Additionally we can use jet particles to control fluid flow during a simulation

by using them as velocity constraints.

Flows can be created easily and quickly by generating low resolution simulations. However there is no easy way to map the parameters used to the higher resolution grids in order to get the same quantitative results. One approach for smoke is to couple the simulation with the low frequencies of a coarse or art directed flow, whilst adding new higher frequencies during the simulation [25].

Instead of using the entire coarse flow to drive a new simulation, only the most significant topological structures may first be extracted from the flow field. The Lagrangian Coherent Structure field, representing the dynamic fluid skeleton of a flow, can be used to create a force that targets the simulated smoke velocity to a low resolution flow, but only in this critical region [26]. For liquid simulations, volumes carrying velocity information can be extracted from a low resolution simulation to guide the flow, whilst simulating the finer details [27]. Alternatively artists may control fluid simulations more directly, using a technique to control the flux through boundaries given by an animated surface [28].

Von Funck, Theisel, and Seidel [29] construct divergence-free vector fields in three dimensions which they use to create an application for sculpting and editing shapes, but they do not construct fluid-like flows for editing.

We will show how to sketch fluid-like flows with jet particles in Section 4.4, and how to add turbulence-like effects to existing flows whilst implicitly enforcing the incompressibility condition in Section 5.4.

2.5 Model Reduction

The degrees of freedom of the solution to an incompressible fluid solver can be significantly reduced by calculating only the most significant bases for the velocities in the domain. This is an example of model reduction, a technique to find solutions on the full space of a system using a lower dimensional basis.

Using dimension reduction, Treuille, Lewis, and Popović [30] create a lower dimensional space to represent the velocity vectors in the fluid domain, and use Galerkin projection to find reduced operators for the fluid evolution equations. This

allows a real-time interactive fluid simulation system, albeit at the cost of a loss of accuracy.

The authors first create a set of example velocity vectors in a matrix U from snapshots of off-line simulations. By construction these velocities are divergence-free and satisfy the no-slip boundary conditions, as constraints of the simulation. To construct the basis velocity vectors, only the most significant eigenvectors are retained from UU^T, corresponding to a principal component analysis. This gives velocity vectors which satisfy the same velocity constraints as above.

Moving objects through the fluid domain represents a useful method of user interaction with the flow. This is made possible in this setting by calculating a reduced basis for the space of velocities that may be observed due to the presence of a boundary object, and adding forces that enforce the boundary conditions at the surface.

De Witt, Lessig, and Fiume [9] develop another method for the Galerkin projection of the Navier-Stokes equations using a spectral basis for the velocity. This basis is calculated using eigenfunctions of the Laplacian operator on the fluid domain.

These eigenvectors are orthogonal, and larger eigenvalues correspond to velocity fields with smaller vortices. By truncating the range of eigenfunctions used, it is possible to make a trade off between accuracy and efficiency by discarding the velocity fields containing smaller vortices. Further, the structure coefficients used for performing the advection are pre-calculated for efficiency. Moving obstacles may be incorporated into this system without calculating an additional reduced basis for the boundary velocities as in the previous method proposed by Treuille, Lewis, and Popović [30]. This method works on regular grids or arbitrary meshes, and does not require example velocity fields from simulation data to create its basis vectors.

Liu et al. [31] present a variational fluid integrator that extends the work of De Witt, Lessig, and Fiume [9]. This method works with arbitrary embedded boundaries even when the domain is a regular grid [31] and gives improved numerical results. The proposed variational integrator preserves energy and displays Kelvin's

circulation theorem.

The above approaches use basis vectors that are global and cover the whole domain, but model reduction can also be implemented using a more local approach. By creating areas which capture local fluid behaviour, Wicke, Stanton, and Treuille [32] cover the fluid domain with tiles, each one containing their own velocity bases. They introduce constraint reduction to modify the basis vectors so that velocities match between adjacent tiles. However their constraints do not guarantee smooth velocity fields across boundaries, and allow discontinuities tangential to the boundaries.

Kim and Delaney [33] develop a subspace integration method that is able to provide variations of previously calculated simulations. This allows efficient re-simulations of the flow with modified parameters from the original. It uses a cubature approach to compute the advection in the reduced space, requiring the pre-computation of cubature points based on previous simulations.

Weißmann, Pinkall, and Schröder [34] propose a method for extracting vortex filaments from existing flows. This method has applications for representing the fluid as a sparse data set, and for providing an input for hybrid solvers that track vortex filaments. Improving on this technique, a hierarchical representation of a flow using vortex filaments can be calculated which represents the flow at multiple scales [35].

Later we shall see that the flow by jet particles approximates Euler-α flow under the assumption that the particle spacing remains close to the kernel size of the particle [36]. So jet particle flow can also be viewed as a model reduction since the high dimensional solution space is reduced and instead represented with a low dimensional basis given by the particles and their state.

Further, jet particle flow is Hamiltonian and therefore reversible and energy preserving.

2.6 Turbulence

Incorporating turbulence effects into fluid dynamics solvers is important in creating convincing and realistic simulations.

Bridson, Houriham, and Nordenstam [37] show how to create procedural divergence-free noise by calculating the curl of a vector-valued 3D noise function [37]. Any continuous function can be used as input, and Perlin noise [38] is a good candidate as it is easy to compute and to control the frequencies in the noise.

Jet particle simulations also create divergence-free velocity fields. Given any random configuration of jet particles and momenta that we choose, the jet particle velocity field can similarly be interpreted as divergence-free noise. However as we shall see in Section 3.6 they can also provide a time varying noise function, by adding the dynamics given by their equations of motion.

Regions of vorticity advected in the fluid contribute to the dynamics of the flow, and have a major role to play in turbulence effects. The numerical dissipation of vorticity in the Euler equations leads to artefacts in its solution. This dissipation is addressed using vorticity confinement [39], by adding a new confinement term to the Euler equations. The modification allows the preservation of vortical regions in the flow even when diffusion is present, but it can lead to the unstable amplification of vorticity in some cases.

Vorticity confinement is improved by coupling the flow with vortex particles [40]. Each vortex particle stores a vorticity value, and a vorticity field is interpolated from them by summing the contributions from nearby particles. These particles incorporate back the vorticity at larger scales than the vorticity confinement, and are used to generate turbulence effects in smoke simulations.

In order to enhance the fluid re-scaling process, Kim et al. [41] add divergence-free wavelet noise during the re-simulation of a lower resolution flow. The noise is modelled according to the Kolmogorov turbulence spectrum [42], which presents a model of how the energy behaves as it diffuses into smaller wavelengths. This approach generates physically plausible turbulence at the sub-grid level that is missing

from the input flow. The corresponding divergence-free noise field is advected with the flow in order to augment the velocity field. This technique attracted a Technical Achievement Award from the Academy of Motion Picture Arts and Sciences in 2012.

Wavelet turbulence can be also be attached to particles [43] in order to augment lower resolution fluid simulations. In this work the Launder and Sharma [44] energy and dissipation model tracks the turbulence on the particles, which are advected with the flow in order to synthesise additional turbulent noise, which is used as a displacement offset during rendering.

Turbulence effects can also be added to existing water simulations by creating wave simulations on liquid surfaces [45]. This idea was further extended to work directly with particle based simulations without requiring a level set [46].

Jet particles may also have a significant role in adding turbulent detail at finer scales to fluid simulations.

2.7 Fluid Editing

The ability to edit and manipulate a fluid simulation in visual effects is important to animators and supervisors trying to achieve a precise vision of their ideas. This is a hard problem to solve due to the infinite solution space and the chaotic nature of fluid dynamics.

Editing a flow given by particles is addressed in Pighin, Cohen, and Shah [47] using a deformation defined with radial basis functions that is advected with the fluid. This method allows local edits to be propagated to nearby particles for a window in time using hierarchical B-splines, although the method used for interpolation of the edits is not physically based.

Smoke simulations that hit predefined keyframes can be generated using control forces and non-linear optimisation methods [48], but this involves an expensive forward gradient computation.

Editing incompressible fluids often requires the design of local control forces or velocity fields that are divergence-free for use in constraints. Treuille et al. [48]

provide a method to create Gaussian wind and vortex forces that target the fluid towards specific key frames. However these vector fields have to be made divergence-free in an additional projection step. We will use divergence-free matrix kernels that are intrinsically divergence-and free to generate our flows.

The simulation targeting method of Treuille et al. [48] is later improved by using the adjoint method, and applied to liquid and smoke simulations [49]. However the method proposed requires a global optimisation which needs to be iterated over the sequence, and they do not address how to create the target keyframes for the simulation.

An alternative method for guiding liquid simulations uses proportional derivative (PD) control based on user input shapes. This produces good results without requiring expensive non-linear optimisations [50]. Instead a divergence-free external force field and a the gradient of a potential field drive the fluid towards the target shape.

Raveendran et al. [22] propose a similar method for controlling liquids by providing a sparse sequence of control meshes. The meshes are first used to interpolate a volume preserving flow from the input meshes. The velocities calculated from this deformation are then used as boundary condtions on the grid during simulation in order to achieve the constrained flow without using PD control.

Zhang et al. [51] describe a technique for controlling particle-based fluid simulations without requiring a grid based solver. Their method firstly generates a particle representation of the input shapes, and then creates a corresponding skeleton motion using joints and weights for each particle. However the system requires the user to provide the joint information as an input, which may not always be available.

A method for interactive editing of liquid simulations is described by Pan et al. [52] that involves sketching an edit and propagating this into the flow whilst simulating the new result. The input to the editing system is a set of points and their edited positions. From this a divergence-free flow is generated by non-linear optimisation of an objective function. This deformation is applied incrementally over previous frames before the edit is incorporated as a constraint during a re-

simulation.

Our editing application differs from this approach as it allows realistic variations to be produced without a re-simulation. Moreover our edits do not need to be made divergence-free using an optimisation step.

The hierarchical vorticity skeletons calculated by Eberhardt et al. [35] can be used for editing the flow field of a single frame. They demonstrate this by means of a spline that translates and rotates all neighbouring filaments, from which a new velocity field can be calculated. However this technique does not guarantee temporal coherence of the vortex filament structures, and so is not well suited for edits over time. In contrast, our technique for editing the fluid flow allows edits to propagate to neighbouring frames.

2.8 Space-time Methods

Space-time methods for fluids have become popular in recent years, taking advantage of the fact that patterns reoccur at different times and positions within the fluid.

Raveendran et al. [53] use space-time interpolation to create blends between pre-calculated liquid simulations using mesh surfaces extracted from the flow. However, the system relies on blending qualitatively similar features that have been picked from each flow by hand. This technique is improved by performing matching automatically using a five-dimensional optical flow technique [54]. The method targets signed distance functions in order to interpolate between both liquid and smoke simulations in space time.

Pan and Manocha [55] combine a space-time approach with optimal control forces in an improved technique for guiding smoke simulations between keyframes of density and velocity fields.

A system for sculpting liquid animations given a number of input meshes [56] provides editing tools for combining space-time features. Liquid droplets are spliced into an existing simulation after applying basic linear transformations.

Chu and Thuerey [57] adapt an algorithm developed from machine learning to synthesise high resolution flow based on repositories of space-time simulation data

using convolutional neural networks.

2.9 LDDMM

The problem of finding correspondences between shapes occurs frequently in the fields of computer vision, for example in optical flow and medical imaging. However a complication in finding a registration between image data using displacement fields is its failure when the deformation required is large.

The large deformation diffeomorphic metric mapping framework (LDDMM) approaches the non-linearity of this deformation space by generating the flow using particle methods [58]. These methods generate a deformation by combining linear combinations of basis functions defined around each particle. A variation of this framework, the large deformation diffeomorphic *kernel bundle* mapping (LDDKBM) framework [59], provide a registration algorithm that allows deformation at multiple scales by varying the particle kernel size, and the authors provide an optimised implementation using the fast multipole method [60].

The related problem of finding a flow that transforms N particles or *landmarks* from one configuration into another [61] gives rise to a Hamiltonian which is a function of the particle's momenta and a choice of kernel [62]. The equations of motion for this Hamiltonian can be used to create incompressible fluid flows if the kernel is chosen to be divergence-free.

By defining a metric given by particle's flow, distances between shapes can be calculated by finding the geodesic flow that transforms one set of points, or one surface, onto another [63]. We use this metric to provide an estimate of the size of the flows we sketch with our fluid editing tools.

The translation and rotation invariant (TRI) matrix kernels define a family of localised vector fields that can vary smoothly between scalar Gaussian, divergence-free or curl-free [62]. These matrix kernels provide the basis for our editing tools by enforcing incompressibility easily, and the ability to soften the incompressibility constraint as required is useful for more creative control.

2.10 Jet Particles

An alternative description of incompressible fluid flow is given by Arnold [64] in his famous paper in 1966. Arnold [64] shows that the flow of an ideal incompressible fluid is equivalent to motion along a geodesic curve in the Lie group SDiff of smooth volume preserving diffeomorphisms on the fluid domain. This is a fascinating viewpoint which links the physics of the fluid to the geometry of the flow.

The Lie algebra $\mathfrak{sl}$, defined as the tangent space to the group SDiff at the identity, consists of the vector space of divergence-free vector fields. These vector fields are the infinitesimal generators of the group SDiff. Although SDiff is infinite, its discretisation [65] leads to an Eulerian integrator for incompressible fluid flow [66] that preserves energy, is unconditionally stable, has no numerical viscosity, and for which Kelvin's circulation theorem holds.

On the other hand, Lagrangian descriptions of Eulerian flow may be derived using Lagrange-Poincaré reduction, where the particle relabelling symmetry allows the equations of motion of an ideal fluid to be projected onto a finite set of particles [67]. Cotter et al. [68] derive *jet particles* as one such reduction, yielding Hamiltonian particles which carry extra information about the deformation gradient of the flow with them. Further, these jet particles represent a Lagrangian discretisation of the diffeomorphism group SDiff [69], and induce a divergence-free vector field in which the particles are carried.

We will see in Section 3.6 that jet particles move in velocity fields that are intrinsically divergence-free, without requiring a pressure solve. The link between the particles momenta and the vector field that generates the flow is established using matrix-valued kernels [36, 62]. The vector fields they induce are divergence-free, and hence generate incompressible flow. This contrasts for example with SPH simulations, where the velocity fields are not divergence-free and the simulations generate compressible flow.

Using ideas from geometric mechanics [70], Sommer and Jacobs [71] examine how jet particle flow and LDDMM methods for geometry matching are both examples of the use of reduction by Lie group symmetry to generate particle meth-

ods. Moreover, jet particles that carry second order information can also be used to construct flows that match images to higher order spatial accuracy [72].

The Euler-Poincaré equations (EPDiff) [70] are used in medical imaging to estimate the geodesic distance of the flow that matches templates between different medical scans [73]. See Miller, Trouvé, and Younes [74] for an approach to finding such optimal deformations in computational anatomy using geodesic shooting. In fact Euler's equations of incompressible flow may be recognised as EPDiff$_{Vol}$, the incompressible version of EPDiff [36]. Since jet particles are singular solutions to the EPDiff equations, they are well suited to our application of sketching and editing incompressible flows.

The connection between jet particle flow and the regularised Euler equations is further explored in Cotter et al. [69]. They show simulations of collisions between two 0-jet particles that result in the creation of a new 1-jet particle, and they provide illustrations of the vector fields given by 2-jet particles. Jet particle simulations have an issue with stability when the particles become too close [36]. We discuss this later and show some stabilising constraints in Sections 4.3 and 4.5.

We are guaranteed the existence of geodesics connecting nearby diffeomorphisms for EPDiff [75], a theorem from Riemannian geometry. So in particular we can connect edited flows to the original ones with a minimal smooth flow. We will construct such deformations with jet particles.

It has been suggested that jet particles can be used to approximate velocity fields that interpolate known velocities at their point positions for metrology [76] but a method for doing so is not provided. We describe an implementation for this in the case of divergence-free velocity fields in Chapter 6.

2.11 Vector Field Design

Designing vector fields is a problem that occurs frequently in computer graphics in two and three dimensional spaces. The applications include fluid dynamics, texture synthesis and non-photorealistic rendering.

For example Zhang, Mischaikow, and Turk [77] create a system for designing

vector fields by placing singularities on surfaces.

In the vector field interpolation problem, vectors are specified at a sparse set of vertices, and these vectors are interpolated over the mesh. This is addressed for tangent fields by interactively specifying tangent vectors at sparse vertices on a triangle mesh using discrete external calculus [78].

Generalising vector fields to n-rosy fields [79] and n-direction fields [80] on surfaces is also important, in particular when n is 4 for re-meshing and parameterisation. Direction fields can also be constructed on discrete surfaces by placing isolated singularities and constructing a trivial connection [81] by solving a sparse linear system.

These are relevant since we can interpolate vector fields given by a set of jet particles at sparse positions in two and three dimensions to create divergence-free vector fields. This is of particular interest in fluid sketching, simulation and editing.

Chapter 3

Technical Background

In this chapter we introduce the technical background used in this thesis, including matrix kernels, hamiltonian dynamics and jet particles.

3.1 Nomenclature

Table 3.1 shows a list of common symbols used in this thesis and their meaning.

Symbol	Meaning	Representation
d	Dimension of domain	$d \in \mathbb{N}$
n	Number or particles	$n \in \mathbb{N}$
M	Domain of motion	Manifold $M \subset \mathbb{R}^d$
N	Number of time steps	$N \in \mathbb{N}$
i, j, k, l, m	Tensor indices	$i, j, k, l, m \in \{1, \ldots, d\}$
α, β	Particle labels	$\alpha, \beta \in \{1, \ldots, n\}$
$\mathbb{I}_d$	Identity matrix	Identity matrix of size $d \times d$
δ^{ij}	Kronecker delta	$\delta^{ij} := [i = j]$
K^{ij}	Matrix kernel tensor	Rank 2 tensor in d dimensions
$\mathrm{SDiff}(\mathbb{R}^d)$	Ideal fluid configuration space	Group of volume preserving flows
$\mathfrak{X}_{\mathrm{div}}(\mathbb{R}^d)$	Divergence-free vector fields	Vector $u \in \mathbb{R}^d$
$\mathrm{SL}(d;\mathbb{R})$	Special linear group in $\mathbb{R}^d$	Matrix of size $d \times d$ with det 1
$\mathfrak{sl}_d(\mathbb{R})$	Lie algebra to $\mathrm{SL}(d;\mathbb{R})$	Matrix of size $d \times d$ with trace 0
$\varphi(x,t)$	Flow function	Function $\varphi(x,t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$
X_i	Discrete grid coordinate	Position $X_i \in \mathbb{R}^d$
D	Density grid	Function $D : X_i \to \mathbb{R}$
U	Velocity grid	Function $U : X_i \to \mathbb{R}^d$
Φ	Deform grid	Function $\Phi : X_i \to \mathbb{R}^d \times \mathbb{R}^{dxd}$

Table 3.1: An explanation of the symbols used throughout this document

We will use the Einstein summation convention, where repeated indices imply summation over the index range.

3.2 Basic Definitions

It is assumed that the reader is familiar with the basic concept of manifolds. However in this thesis it is sufficient to consider a manifold M to be the same as the space $\mathbb{R}^d$, where the dimension d is either two or three. The interested reader who would like to know more should refer to Abraham, Marsden, and Ratiu [82] for a good introduction into the theory.

We use the following definitions frequently in this thesis:

Definition 3.1. Let M_1 and M_2 be two manifolds. A **diffeomorphism** is a continuous bijective map $\varphi : M_1 \to M_2$ whose inverse φ^{-1} is also continuous.

Definition 3.2. Let $\varphi : M_1 \to M_2$ be a map between manifolds with $f : M_2 \to \mathbb{R}$ a function on M_2. Then the **pull-back** of f by φ is a function $\varphi^* f : M_1 \to \mathbb{R}$ on M_1 given by

$$\varphi^* f = f \circ \varphi$$

Definition 3.3. Let $f : M_1 \to M_2$ be a C^1 continuous maps between two manifolds. The **tangent** Tf of f is the map

$$Tf([c]_x) = [f \circ c]_{f(x)}$$

where $[c]_x$ is the equivalence class of curves $c : [a,b] \to M_1$ that have the same tangent at $x \in M_1$.

Definition 3.4. Let $\varphi : M_1 \to M_2$ be a *diffeomorphism* map between manifolds and $U \in \mathfrak{X}(M_1)$ a vector field in M_1. Then the **push-forward** of U is a vector field $\varphi_* U \in \mathfrak{X}(M_2)$ in M_2 defined by

$$\varphi_* U = T\varphi \circ U \circ \varphi^{-1}$$

We shall use the push-forward of vectors later in order to calculate the image of velocity vectors under a deformation.

Definition 3.5. Let $\mathfrak{X}(\mathbb{R}^d)$ denote the space of vector fields on $\mathbb{R}^d$ and let $u(t) \in \mathfrak{X}(\mathbb{R}^d)$ be a time-dependent vector field where $t \in [0,1]$. Then the **flow** of u is the time-dependent diffeomorphism $\varphi_t^u : \mathbb{R}^d \to \mathbb{R}^d$ defined by

$$\varphi_t^u(x_0) = x(t)$$

where $x(t)$ denotes the solution to the initial value problem

$$\frac{dx}{dt} = u(x) \quad x(0) = x_0$$

If we restrict $\mathfrak{X}_{\text{div}}(\mathbb{R}^d)$ to the space of *divergence-free* vector fields then the flow φ_t^u is *volume preserving*.

We denote the space of volume preserving diffeomorphisms by $\text{SDiff}(\mathbb{R}^d)$, which under reasonable circumstances is a smooth infinite dimensional manifold.

Definition 3.6. The Lie derivative of a tensor field defines the change in the tensor along the flow of a given vector field. Let X be a vector field, Y be a tensor field and φ_t^X be the *flow* of X. Then the **Lie derivative** of Y with respect to X is

$$\mathscr{L}_X Y(p) = \lim_{t\to 0} \frac{(\varphi_{-t}^X)_* Y(\varphi_t^X(p)) - Y(p)}{t}$$

3.3 Introduction

Key to our method is the use of particles to design incompressible flows. In addition, we seek deformations that are "fluid-like" in some sense.

It was shown by Arnold [64] that the solutions to the ideal fluid equations are in fact geodesics on SDiff, the group of volume preserving diffeomorphisms, for the right invariant L^2 metric. Motivated by this, a number of regularised fluid models have been proposed by giving SDiff stronger Riemannian metrics induced by Sobelev norms [83, 36]. These models have a regularisation parameter σ, such

that its solutions converge to the ideal fluid equations in the limit, as σ tends towards zero. Therefore when σ is small, their solutions will appear fluid-like over short time-scales.

Most importantly, these regularised models exhibit "particle-like solutions", i.e. solutions obtained by solving a finite-dimensional ODE for the dynamics of the particles. This is a massive reduction, as SDiff is an infinite dimensional Fréchet manifold. The use of such solutions is our primary tool in obtaining deformations which appear to be fluid-like. It is natural to ask how to link the dynamics of finitely many particles with a full diffeomorphism, and matrix-valued kernels provide a key link in this relationship.

3.4 Matrix-valued Kernels

Matrix-valued kernels are well studied in mathematics but have not yet been applied in computer graphics to our knowledge, so we include some relevant technical background in this section.

A matrix-valued kernel is a mapping $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ such that $K(x,y)$ is positive-semidefinite symmetric, and $K(x,x)$ is the identity matrix for all $x,y \in \mathbb{R}^d$. We can use matrix-valued kernels to generate vector fields, given a particle and its momentum.

A particle with position $q \in \mathbb{R}^d$ and momentum $p \in \mathbb{R}^d$ has the state (q,p), and moves in a *phase space* given by $T^*\mathbb{R}^d \equiv \mathbb{R}^d \times \mathbb{R}^d$. A particle at position q then induces a matrix field $K(\cdot,q)$ over $\mathbb{R}^d$. By multiplying the matrix at a point x by the momentum p we produce a vector, and hence we define a vector field $u(x)$ over $\mathbb{R}^d$ when considering all points x. Hence there is a linear map $(q,p) \mapsto u(\cdot)$ from the particle's phase space $T^*\mathbb{R}^d$ to the space of vector fields $\mathfrak{X}(\mathbb{R}^d)$, given by $u(x) = K(x,q) \cdot p$.

This is important for us because there are matrix kernels available that are easy to compute and that generate divergence-free vector fields. In particular if K generates divergence-free vector fields, then any curve $(q(t),p(t)) \in T^*\mathbb{R}^d$ will generate a time-dependent divergence-free vector field $u(x,t) = K(x,q(t)) \cdot p(t)$.

The flow of u is then a volume preserving diffeomorphism (see Definition 3.5).

In this section we will introduce a family of matrix kernels following Micheli and Glauns [62], please refer to the paper for more details. This family of matrix kernels is equipped with a parameter which allows interpolation between a scalar kernel and either a divergence-free or curl-free matrix kernel.

In our application we are interested particularly in kernels which are divergence-free since we are dealing with incompressible flow. However, we mention the family of kernels here since the ability to relax this condition will be useful in our sketching application in Chapter 4.

We will consider here translation and rotation invariant kernels (also called TRI kernels). A matrix-valued kernel is called translation invariant if $K(x+\tau, y+\tau) = K(x,y)$ for any $\tau \in \mathbb{R}^d$. If K is translation invariant, we may write it using an auxiliary function $k : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ which is related to K via $K(x,y) = k(x-y)$. Moreover, we say k is rotationally invariant if $k(Rx) = R^T K(x) R$ for any rotation matrix $R \in \mathrm{SO}(\mathbb{R}, d)$. In this case it can be shown [62] there must exist functions $k^{\parallel} : \mathbb{R}^+ \to \mathbb{R}$ and $k^{\perp} : \mathbb{R}^+ \to \mathbb{R}$ which relate to k via

$$k(x) = k^{\parallel}(\|x\|)\mathrm{Pr}_x^{\parallel} + k^{\perp}(\|x\|)\mathrm{Pr}_x^{\perp}$$

where

$$\mathrm{Pr}_x^{\parallel} = \frac{xx^T}{\|x\|^2}$$

$$\mathrm{Pr}_x^{\perp} = \mathbb{I}_d - \frac{xx^T}{\|x\|^2}$$

for $\|x\| > 0$. Usually the extension to the origin $x = 0$ is achievable by enforcing continuity [62, §3].

For example we may consider the family of kernels

$$k^{\parallel}(r) = be^{-r^2} \qquad k^{\perp}(r) = \left(b - ar^2\right) e^{-r^2}$$

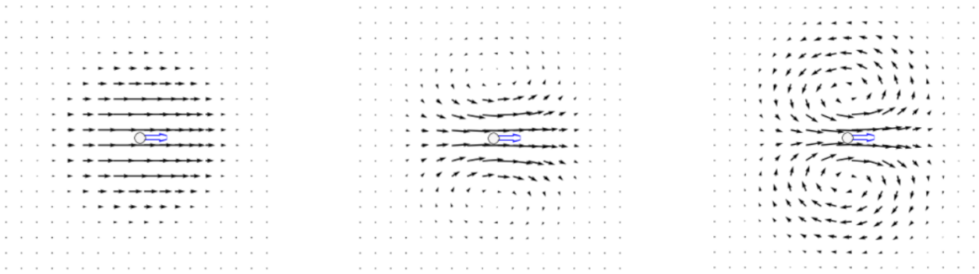


Figure 3.1: The vector fields of the matrix kernel from Equation (3.2) for $a = 0, 0.5, 1.0$ with momenta $(1,0)$ inserted, showing the Gaussian, interpolated and divergence-free kernels from left to right

where $r = \|x\|$ and $a, b \in \mathbb{R}$ which give positive definite kernels for the set

$$D = \left\{(a,b) | b \geq -(d-1)\frac{a}{2c}, a \geq 0\right\}$$

where d is the dimension [62]. Values in D such that

$$b = (d-1)\frac{a}{2c}$$

guarantee that the kernel is divergence-free, so in particular the matrix-valued function

$$K^{ij} = \left(\frac{x^i x^j}{\sigma^2} + \left(1 - \frac{r^2}{\sigma^2}\right)\delta^{ij}\right) e^{-r^2/\sigma^2} \tag{3.1}$$

is a divergence-free kernel with kernel size σ [62]. Here i and j are tensor indices taking values in the range $[1,d]$, and we are using the Einstein convention to indicate that repeated indices are summed over the index range automatically.

By leaving a as a parameter, we have a family of kernels

$$K^{ij} = \left(\frac{a x^i x^j}{\sigma^2} + \left(1 - \frac{a r^2}{\sigma^2}\right)\delta^{ij}\right) e^{-r^2/\sigma^2}. \tag{3.2}$$

which gives the usual Gaussian kernel when $a = 0$, the divergence-free kernel when $a = 1$, and interpolated versions for $a \in (0,1)$ [62]. These kernels are useful for relaxing the condition that the vector field is divergence-free. See Figure 3.1 for an illustration of the vector field given values of $a = 0, 0.5, 1$.

See also Mumford and Michor [36] for a similar divergence-free matrix kernel

and a discussion of its links with Eulerian flow.

3.5 Hamiltonian Dynamics

D'Arcy Thompson's landmark book "On Growth and Form" [84] elegantly demonstrates the link between mathematics and biological processes. It can be argued that the application of the field of computational anatomy to medical imaging is a realisation of the key ideas in this book. It is the use of Hamiltonian control that is fundamental in deriving geodesic flows which drive image registration techniques. See Miller, Trouvé, and Younes [85] for a review of this viewpoint and its application to shape space.

The matrix-valued kernels from the previous section generalise scalar valued kernels that are used to invert differential operators on scalar functions. Specifically, these matrix-valued kernels are used to invert differential operators on vector fields.

Let $\mathscr{A}$ be a pseudo-differential operator and m be the *momentum operator* given by so that $m = \mathscr{A}u(x)$ such that K is the Green's function for the operator $\mathscr{A}$ [68].

Then $u = K * m$ where $*$ denotes the convolution operation

$$u(x) = (K * m)(x) = \int_{\mathbb{R}^d} K(x,y) \cdot m(y) dy$$

and this equation provides the link between the velocity vector field and the momentum of the fluid.

Consider the evolution equation for a divergence-free vector field u given by

$$\partial_t m + \pounds_u m = 0 \quad , \quad m = \mathscr{A}u(x) \tag{3.3}$$

where $\pounds_u$ is the Lie-derivative operator [86].

In the case that $\mathscr{A}$ is the identity transformation, these equations of motion are identical to Euler's equations for an ideal fluid [64, 87].

However in the case that $\mathscr{A} = (1 - \alpha\Delta)$ for $\alpha > 0$ we obtain the Euler-α model of fluid turbulence [88, 83]. And in the case that $\mathscr{A} = (1 - \frac{\eta}{k}\Delta)^k$ we obtain

the model of Mumford and Michor [36].

In any case, the above evolution equation acts as a generalised model for fluid dynamics [36]. If $\mathscr{A}$ admits a kernel K, then the function

$$h(m) = \frac{1}{2}\langle m, K * m\rangle_{L^2}$$

is the Hamiltonian function for this evolution equation [68].

If $\mathscr{A}$ admits a C^k kernel, then Equation (3.3) admits particle like solutions [67]. This represents a key source of computational efficiency, since we can reduce a high dimensional flow to a flow given by finitely many particles. We make use of this observation in Chapter 4 by creating an application for sketching fluid-like flows using constrained particles.

In particular we consider the Hamiltonian function defined on n particles $H : (T^*\mathbb{R}^d)^n \to \mathbb{R}$ given by

$$H(p,q) = \frac{1}{2}K^{ij}(q_\alpha, q_\beta)p_{\alpha i}p_{\beta j} \tag{3.4}$$

where the Hamiltonian is a function of the n particles indexed by particle labels α, β, with each particle having position q_α and momentum p_α, and where K is a matrix-valued kernel. Here i and j are the indices of the matrix K and the momentum p. Note that the term is also summed over α and β, so all particle interactions are included in the Hamiltonian.

The equations of motion of a Hamiltonian system are given by

$$\begin{aligned}\dot{q} &= \frac{\partial H}{\partial p}\\ \dot{p} &= -\frac{\partial H}{\partial q}\end{aligned}$$

and so for this Hamiltonian the equations of motion are

$$\begin{aligned}\dot{q}^i_\alpha &= \quad K^{ij}(q_\alpha, q_\beta)p_{\beta j}\\ \dot{p}_{\alpha i} &= -p_{\alpha j}\partial_i K^{jk}(q_\alpha, q_\beta)p_{\beta k}\end{aligned} \tag{3.5}$$

where ∂_i is the partial derivative $\partial/\partial x_i$.

The key point here is that if the particle motion $(q(t), p(t))$ is a solution of Equation (3.5), then the velocity field $u(x,t) = K(x,q(t)) \cdot p(t)$ is a solution to Equation (3.3) on the full manifold M [36, 67, 68]. Note that $\dot{q}(t) = u(q(t))$, so that if $\varphi(x,t)$ is the flow of u, then $q(t) = \varphi(q(0),t)$ and the particles are carried by this flow.

The velocity field generated by the particles is given by

$$u^i(x) = K^{ij}(q_\alpha, x) p_{\alpha i} \tag{3.6}$$

If K is a divergence-free matrix-kernel such as the one given in Equation (3.1) [68], then the vector field u is divergence-free for all time, by nature of its being a sum of divergence-free vector fields.

The velocity field generated by a single particle using a divergence-free matrix kernel is shown in Figure 3.1 (right).

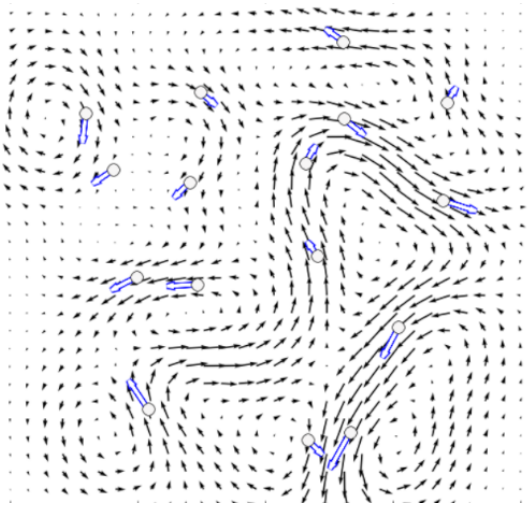


Figure 3.2: Hamiltonian particles. The arrows represent the momenta of the particles.

A collection of particles will interact using the equations of motion in Equation (3.5), and each particle will move in the velocity field implied by the sum of the contributions from all the particles. Figure 3.2 shows the velocity field generated by a number of randomly placed particles with varying momenta.

3.6 Jet Particles

Jet particles extend the dynamics given by the particles seen in the previous section, by carrying extra information that describes the local change to the deformation gradient. To our knowledge they have not been used in computer graphics applications at this time and we give a brief introduction to them here, following Cotter et al. [68].

In particular, jet particles carry information from the truncation of the Taylor series describing the local deformation gradient [68]. Here the term *jet* refers to the jet of a function; a jet is an equivalence class of functions that match a given function up to polynomial order k (see for example Kolár, Slovák, and Michor [89]). And particles that carry information matching the deformation gradient up to order k are named k-jetlets.

Jet particles interact with one another showing dynamics which conserve linear and angular momentum. Their equations of motion are derived using Hamilton's principle [68, 69], and under a suitable choice of Lagrangian the solutions converge to Euler's equations given certain assumptions.

3.6.1 Basic Definitions

The Lagrangian description of an incompressible fluid is given by a time dependent volume preserving map $\varphi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ describing the flow of particles, where d is two or three in this thesis. In particular, the location of a particle at time t with initial position $x_0 \in \mathbb{R}^d$ is given by $\varphi(x_0, t)$.

The deformation gradient at x_0 is the matrix

$$Q^i{}_j := \left.\frac{\partial \varphi^i}{\partial x_j}\right|_{x=x_0}$$

and higher order deformation gradients are defined in a similar way.

In our case $Q^i{}_j$ must lie in $\mathrm{SL}(d,\mathbb{R})$ since the flow is incompressible, where $\mathrm{SL}(d,\mathbb{R})$ is the special linear group of matrices acting on $\mathbb{R}^d$ with unit determinant.

Just as one can obtain solutions to Equation (3.3) by solving for particle trajectories $q(t)$, one can obtain richer solutions of Equations (3.3) by solving for systems

which involve $q(t)$ *and* the deformation gradient $Q(t)$ [68].

These are jet particles, Hamiltonian particles that carry higher order information about the deformation gradient of the flow with them. In the case of n jet particles, the $q(t)$ are in the space $(\mathbb{R}^d)^n$ and the deformation gradients $Q(t)$ are in the space $\mathrm{SL}(d,\mathbb{R})^n$.

3.6.2 Equations of Motion

The phase space is then $T^*(\mathbb{R}^d \times \mathrm{SL}(d,\mathbb{R}))^n$ with canonical coordinates given by (q,Q,p,P) where q denotes particle position, Q denotes the Jacobian matrices, and p and P are their respective conjugate momenta.

Cotter et al. [68] show that the equations of motion for (q,Q,p,P) are Hamiltonian with respect to the Hamiltonian function $H : T^*(\mathbb{R}^d \times \mathrm{SL}(d;\mathbb{R}))^n \to \mathbb{R}$ given by

$$\begin{aligned} H(q,Q,p,P) = \frac{1}{2}\, & p_{\alpha i}\, p_{\beta j}\ K^{ij}(q_\alpha, q_\beta) \\ & + p_{\beta j}\ P_{\alpha i}{}^{\ell} Q_{\alpha}{}^{k}{}_{\ell}\ \partial_k K^{ij}(q_\alpha, q_\beta) \\ & - \frac{1}{2} P_{\alpha i}{}^{m} Q_{\alpha}{}^{n}{}_{m} P_{\beta j}{}^{\ell} Q_{\beta}{}^{k}{}_{\ell}\ \partial_{nk} K^{ij}(q_\alpha, q_\beta) \end{aligned} \tag{3.7}$$

They show that the equations of motion can be simplified by writing them in terms of the variables

$$\mu_{\alpha i}{}^{j} = P_{\alpha i}{}^{k} Q_{\alpha}{}^{j}{}_{k} \in \mathfrak{sl}(d;\mathbb{R})^* \tag{3.8}$$

$$\xi_{\alpha}{}^{i}{}_{j} := \frac{\partial u^i_\alpha}{\partial x^j} = p_{\beta k}\ \partial_j K^{ik}(q_\alpha, q_\beta) - \mu_{\beta \ell}{}^{k}\ \partial_{jk} K^{i\ell}(q_\alpha, q_\beta) \tag{3.9}$$

The variable μ is called the *right-trivialised matrix momenta*, because it is the right trivialisation of the covector $(Q,P) \in T^*\mathrm{SL}(d;\mathbb{R})^n$ over the Lie group $\mathrm{SL}(d;\mathbb{R})$ [68].

Using these new variables, the equations of motion may now be written in terms of $(q_\alpha, p_\alpha, \mu_\alpha)$ [68] as

$$\dot{q}^i_\alpha = p_{\beta j} K^{ij}(q_\alpha, q_\beta) - \mu_{\beta j}{}^{k}\ \partial_k K^{ij}(q_\alpha, q_\beta) \tag{3.10}$$

$$\begin{aligned}\dot{p}_{\alpha i} = &-p_{\alpha k}p_{\beta j}\partial_i K^{kj}(q_\alpha, q_\beta) \\ &+ (p_{\alpha\ell}\mu_{\beta j}{}^{k} - p_{\beta\ell}\mu_{\alpha j}{}^{k})\partial_{ki}K^{\ell j}(q_\alpha, q_\beta) \\ &+ \mu_{\alpha k}{}^{j}\mu_{\beta m}{}^{\ell}\partial_{ij\ell}K^{km}(q_\alpha, q_\beta)\end{aligned} \tag{3.11}$$

$$\dot{\mu}_{\alpha i}{}^{j} \quad = \quad \mu_{\alpha i}{}^{k}\xi_{\alpha}{}^{j}{}_{k} - \mu_{\alpha k}{}^{j}\xi_{\alpha}{}^{k}{}_{i} \tag{3.12}$$

and we can reconstruct the deformation gradient Q_α using the formula

$$\dot{Q}_{\alpha}{}^{i}{}_{j} = \xi_{\alpha}{}^{i}{}_{k}\, Q_{\alpha}{}^{k}{}_{j} \tag{3.13}$$

Note that the vector field $u(x)$ induced by the set of 1-jet particles is given by

$$u^i(x) = K^{ij}(q_\alpha, x)p_{\alpha i} - \mu_{\alpha j}{}^{k}\partial_k K^{ij}(q_\alpha, x) \tag{3.14}$$

and this is also divergence-free [68].

3.6.3 Simulation Example

To illustrate the variety of vector fields that may be produced by 1-jet particles in $\mathbb{R}^2$, we choose the following basis for $\mathfrak{sl}_2(\mathbb{R})$ given by

$$spin = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right),\ stretch = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right),\ shear = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$$

This is not a unique basis for $\mathfrak{sl}_2(\mathbb{R})$, but it is useful to illustrate the behaviour of the vector fields the jet particles can produce.

The velocity field generated by a 1-jet particle with zero momentum p, and μ taking the each of the values *spin*, *stretch* and *shear* are shown in Figure 3.3.

The *spin* particle rotates the fluid locally, the *stretch* particle compresses the particle in the y direction whilst expanding in the x direction, and the *shear* particle creates a velocity field with a shearing flow along the x-axis. Each of these vector fields is divergence-free by construction, and moreover any linear combination of these μ values will generate divergence-free vector fields.

Hence jet particles with varying momentum p and varying μ may be arranged arbitrarily to create divergence-free vector fields. And further, these particles will

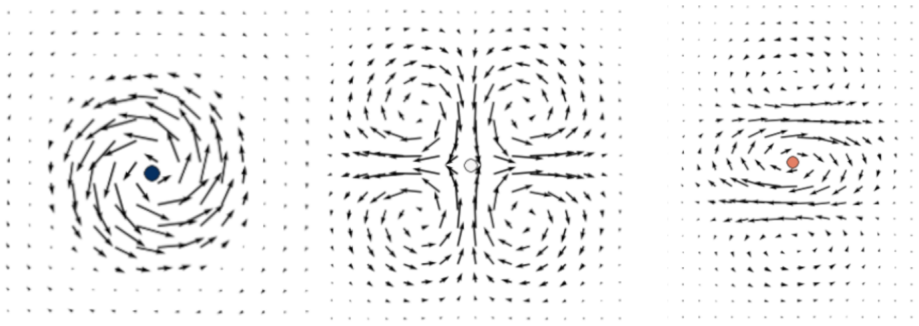


Figure 3.3: A single 1-jet particle in 2D with momentum $p = 0$, and μ taking the values *spin*, *stretch* and *shear*, illustrated from left to right

interact with each other according to the equations of motion (3.10), (3.11) and (3.12), as each particle moves in the summed velocity field given by all the particles given by Equation (3.14).

We demonstrate this with a simple example in two dimensions. We initialise the particles at random positions, and taking random values for p and μ. At each time step the particles state is updated using forwards Euler integration. The algorithm for simulating 1-jet particles using forward Euler integration is shown in Algorithm 1.

Algorithm 1 Jet Particle Simulation

procedure INTEGRATE PARTICLES
    $q_0 \leftarrow$ Initial position of *particles*
    $p_0 \leftarrow$ Initial momentum of *particles*
    $\mu_0 \leftarrow$ Initial shape *particles*
    $t \leftarrow 0$
loop $t < t_{\max}$:
    $q_{i+1} \leftarrow q_i + \dot{q}_i\,dt$ Using Equation (3.10)
    $p_{i+1} \leftarrow p_i + \dot{p}_i\,dt$ Using Equation (3.11)
    $\mu_{i+1} \leftarrow \mu_i + \dot{\mu}_i\,dt$ Using Equation (3.12)
    $t \leftarrow t + dt$.

Figure 3.4 shows the velocity field generated by a number of randomly placed particles $(q_\alpha, p_\alpha, \mu_\alpha)$ with varying momenta p and varying μ. The full simulation is shown in Video 4.6.

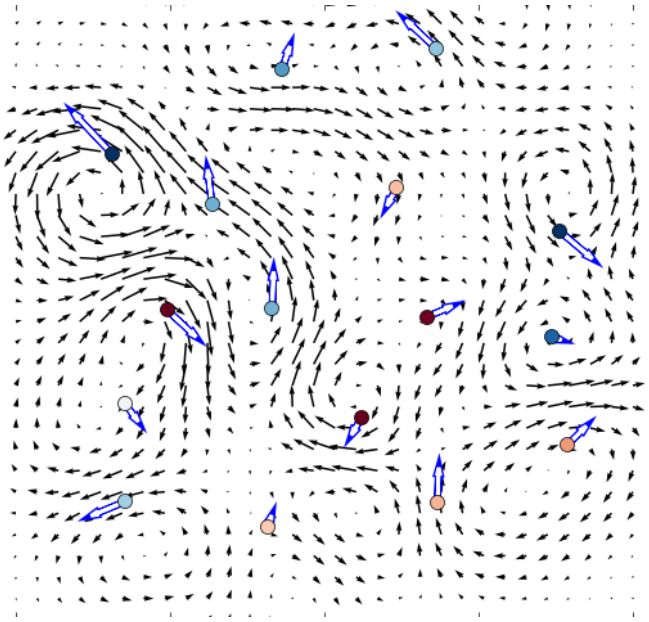


Figure 3.4: The velocity field generated by several 1-Jet particles. The colours of the particles represent the vorticity of the particles, with red and blue being positive and negative vorticity.

3.7 Relationship with Eulerian Flow

In this section we motivate the use of jet particles for the sketching of fluid-like motion by reviewing their links with Eulerian flow.

Using the Lagrangian given by the kinetic energy of the fluid

$$\mathscr{L}(u) = \|u\|_{L^2}^2 = \int |u|^2 d^n x$$

then by Hamilton's principle

$$\delta \int \mathscr{L}(u) dt = 0$$

it can be shown that the equations of motions correspond to Euler's equations

$$\begin{cases} u_t + (u \cdot \nabla)u + \nabla p = 0 \\ \nabla \cdot u = 0 \end{cases}$$

See Cotter and Holm [90] for the derivation of Euler's equations using the Clebsch

variational principle.

If instead we consider the regularized Lagrangian given by

$$\mathscr{L}(u) = \frac{1}{2}\int u \cdot \mathscr{A}u \, d^n x$$

where

$$\mathscr{A} = \left(1 - \frac{\sigma^2}{k}\nabla\right)^k \left(1 - \frac{1}{\varepsilon}^2 \nabla \circ \mathrm{div}\right)$$

then it can be shown that the application of Hamilton's principle gives the EPDiff equations

$$\begin{cases} \partial_t m + u \cdot \nabla m + (\nabla u)^T \cdot m + m(\mathrm{div} u) = 0 \\ u = K * m \end{cases} \tag{3.15}$$

where u is the convolution of the Green's function K for the operator $\mathscr{A}$ and $m(x) := \mathscr{A}u(x)$ is the *momentum operator* [68]. See Cotter et al. [68] and references therein for more details.

In fact, the Euler equations are recovered in the limit of these more regularised EPDiff equations [36]. In particular, in the limit as ε and σ go to zero, the solution to these equations converge to the incompressible Euler equations [68]. Moreover, when $\varepsilon = 0$ and $k = 1$ we recover the regularised Euler-α equations [87] for incompressible fluids, where the activity is filtered below a length scale α.

Hence under certain assumptions, jet particle solutions can be considered solutions of Euler's fluid equations. However, instabilities are observed in jet particle simulations that lead to particle collisions [36], which mean they cannot be used reliably to simulate fluid flow without modification.

Despite the presence of these instabilities, jet particles are suitable for generating divergence-free fluid flows and are natural for controlling fluid simulations, as we will demonstrate in this thesis. We go on to demonstrate their use in the generation of flows via sketching in the next chapter.

Chapter 4

Sketching Flows with Jet Particles

Fluid simulations are hard to control, show chaotic behaviour with sensitivity to initial conditions and are expensive to compute. It is for these reasons that designing a fluid simulation is a difficult problem. As such, the ability to sketch flows easily is an important technique in computer graphics. One relevant application of this is the rapid prototyping of low resolution flows to be used as input for high resolution fluid simulations [41].

Transforming a hand movement into a stable and realistic flow in an efficient manner is a non-trivial problem. In this chapter we describe how to sketch such flows in a simple fashion, using jet particles that are constrained to curves created by an artist. We demonstrate the sketching technique in two dimensions and describe how to extend it to three dimensions.

4.1 Sketching Flows Interactively

We seek to design a deformation that could have been the result of a fluid flow, and to do this efficiently without resorting to the use of a fluid simulator. A natural way to proceed is to take the artist's mouse movement as the input to this system, or equivalently a set of parameterised curves that have been designed for the purpose.

An important feature in our sketching application is that there should be a minimal and sufficient number of controls to achieve the desired effect. Since the space of possible output configurations of a fluid simulation is infinite, reducing the parameter space to a finite number of particles gives a more practical and intuitive

way to achieve our goal.

In our method we propose using a constrained Hamiltonian system to design a deformation that is smooth, divergence-free and globally injective (so it is guaranteed not to fold). By recording a path of the artist's input we define a space curve representing her brush stroke, and in this section we use this as a constraint for a jet particle system.

As we have seen in Chapter 3.5, jet particle systems defines a vector field which is divergence-free, and that can be integrated to generate a deformation which will have a fluid-like motion. We show how to propagate this deformation into the flow and how to apply it to a fluid cache in Chapter 5.

4.2 Deformation with Constrained Particles

We first consider how to create a constrained flow using 0-jet particle dynamics in two dimensions, and we show that the generalisation to three dimensions is trivial.

In our system, the artist selects a brush size for the edit by specifying a value for σ to be used in the matrix kernel. We will interpret the brush stroke as a space curve $\gamma : I \to \mathbb{R}^d$ given by $s \mapsto (x(s), y(s))$, where $\gamma(s)$ is parameterised over an edit time interval $I = [0, S]$.

We will consider this curve to represent the dynamics of a reference particle with state $(\bar{q}(s), \bar{p}(s)) := (\gamma(s), \frac{d\gamma}{ds})$. Of course a single particle without constraints will travel in a straight line conserving momentum, but in contrast the artist's brush stroke is an arbitrary curve. Hence instead we may view the motion of $\bar{q}(s)$ to be the constrained motion of a particle under external forces $F(s)$ that provide the change in its momentum.

By applying a constraint implied by the artists input, we can override the dynamics of the Hamiltonian to force q onto $\bar{q}$. However we can also relax this constraint by providing a maximal force $F_{\max}$ that is allowed. In this sense, the brush stroke becomes a guide for the particle, and the motion of the particle is now viewed as a constrained Hamiltonian system.

To proceed we will first update the particle with the equations of motion and

then apply the impulse from the constraint.

4.3 Constrained 0-Jet Particles

As the artist paints with their input device, we record the positions of the brush, sampling at intervals of δs at times $s_i := i\delta s$. We store these positions as a discrete curve $\Gamma = \{X_1, \ldots, X_N\}$ with points $X_i = \gamma(s_i) = (x(s_i), y(s_i))$.

To initialise the dynamics of the particles we define the initial conditions to be $q_0 = X_0$ and $p_0 = 0$. In order to implement a hard constraint where the jet particle q is exactly constrained to the brush particle $\bar{q}$, we add an impulse J_i to the system which acts over the edit time interval $[s_i, s_{i+1}]$. An impulse is a change of momentum, and here it is precisely the change of momentum required to match the velocity of the artist's curve.

To achieve this constraint, we modify the integration steps as shown in Algorithm 2 with the addition of the impulse J_i at each time step. This ensures that the simulated particle q will exactly match the brush particle $\bar{q}$.

Algorithm 2 Constrained Particle Simulation

1: **procedure** INTEGRATE PARTICLES
2: $q_0 \leftarrow$ initial brush position $\bar{q}_0$
3: $p_0 \leftarrow$ initial brush momentum $\bar{p}_0$
4: $s \leftarrow 0$
5: *loop* $s < s_{\max}$:
6: $q_{i+1} \leftarrow q_i + \dot{q}_i \delta s$
7: $p_{i+1} \leftarrow p_i + \dot{p}_i \delta s + J_i$
8: $s \leftarrow s + \delta s$

To achieve the full constraint we set $J_i = \bar{p}_i - \bar{p}_{i-1}$, while in order to relax the constraint we clamp J_i such that $J_i/\delta s < F_{\max}$.

To provide an interface for the artist where we do not need to specify $F_{\max}$, we choose to interpolate linearly between the p_i and $\bar{p}_i$ at each time step using a blending parameter $b \in [0,1]$. This has the effect of specifying how much to blend the target particle's momentum into the simulated particle:

$$p_{i+1} \leftarrow (1-b)(p_i + \dot{p}_i \delta s) + b\bar{p}_i$$

From this point onwards we use this latter approach of blending the momenta, since the blending parameter is more intuitive than the maximal force parameter, being a scale independent parameter in the range $[0,1]$. We use this technique in Section 4.7 in order to stabilise the dynamics of multiple jet particles.

4.4 Fluid Sketching Application

We have created a fluid sketching application to allow the creation of flows with a 0-jet particle using the above techniques. To illustrate the sketching algorithm we show how our application uses a constrained 0-jet particle to distort a grid.

The results are shown in Videos 4.1, 4.2 and 4.3.

Video 4.1 (Sketching with a scalar kernel)**.** This shows the baseline flow that results in Figure 4.1 using the scalar kernel. This flow is not incompressible and does not use the matrix kernel.

Video 4.2 (Sketching with a divergence-free matrix kernel)**.** Our application generates an incompressible flow that results in Figure 4.2, by constraining a 0-jet particle to a user-generated curve sketched with a mouse.

Video 4.3 (Sketching with an interpolated matrix kernel)**.** The result of using an interpolated matrix kernel with a value of $a = 0.5$ is shown in Figure 4.3. The flow is not incompressible, but may useful for stylistic effect.

In each case the deformation is designed using a mouse, by sketching an arc in a simple clockwise movement from top to bottom.

Note that by using Equation (3.2) and retaining the parameter $a \in [0,1]$ when evaluating our matrix kernel, we are able to interpolate between scalar and divergence-free deformations which will give the artist more creative control. The result of using a value of $a = 0.5$ is shown in Figure 4.3.

We believe that this parameter will allow the user to change the stylistic effect of the flow by altering the incompressibility in this way.

This application is a key component of our fluid editing system and we will also use it later in Chapter 5 to generate keyframes for editing existing simulations.

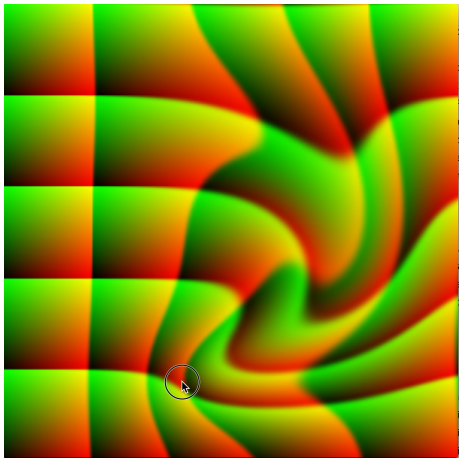


Figure 4.1: A constrained particle with a scalar kernel sketching a flow deforming a grid

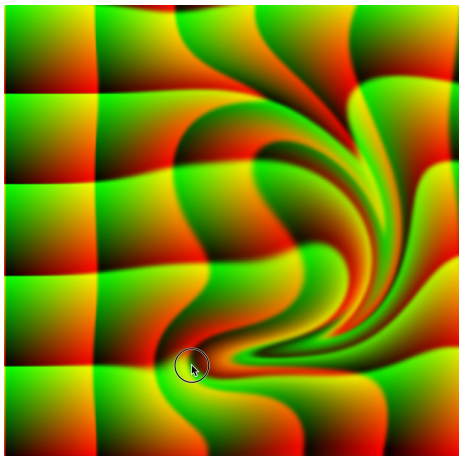


Figure 4.2: A constrained 0-jet particle sketching a flow deforming a grid

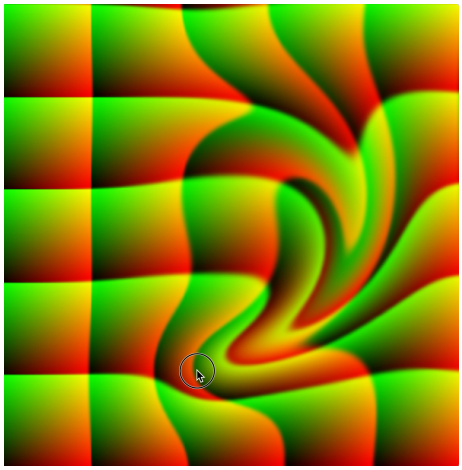


Figure 4.3: A constrained 0-jet particle sketching a flow deforming a grid with $a = 0.5$

This technique is easily extended to three dimensions without complications. If we now evaluate the velocity field with the same matrix kernel

$$u^i(x) = K^{ij}(q_\alpha, x)p_{\alpha i} \qquad \text{(3.6 revisited)}$$

in three dimensions, we still generate a divergence-free velocity. And as above we can choose a brush size σ, and now constrain a 0-jet particle to an artist's curve defined in 3D.

The velocity field for a 0-jet particle in 3D is shown in Figure 4.4, with the streamlines illustrated in three orthogonal views.

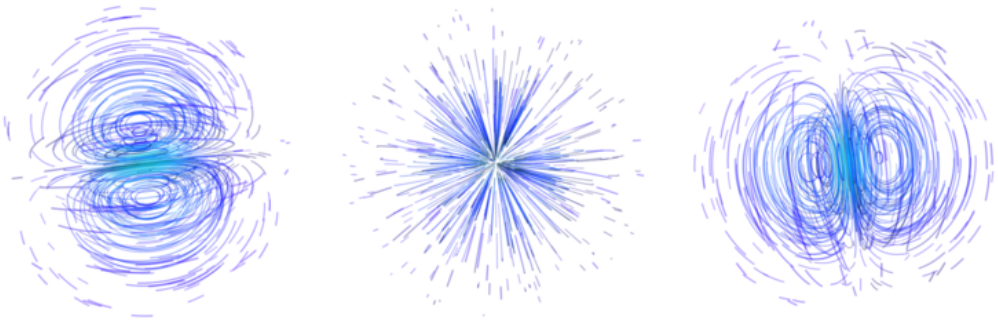


Figure 4.4: Streamlines of 0-jet particles in 3D with momenta along the x, y and z axes, from left to right.

We use jet particle flow using these 3D matrix kernels in Chapter 5 to deform existing flows.

4.5 Constrained 1-Jet Particles

If we want to design a flow which can locally rotate and scale the fluid as well as translate it, we can use 1-jet particles to modify the deformation gradient at each particle location. We will specify a target $\bar{\mu} \in \mathfrak{sl}_d(\mathbb{R})$ as a constraint for the 1-jet particle dynamics to control the deformation. In this section we will either choose our target $\bar{\mu}$ to be constant, or we may choose to align its matrix with the brush stroke direction for more intuitive control.

In order to align a particular brush shape with the painting direction, we choose a rest shape $\bar{\mu}_0 \in \mathfrak{sl}_d(\mathbb{R})$, which we choose in a local coordinate system. Then

we find a rotation matrix R which aligns it to the direction given by the particle's momentum p, so that $R(e_x) = p$ where e_x is the unit vector in the x direction.

Now if we define our target $\bar{\mu}_i$ such that

$$\bar{\mu}_i = R\bar{\mu}_0 R^T$$

then $\bar{\mu}_i$ will be oriented correctly with respect to the brush direction. The matrix $\bar{\mu}_i$ is still in $\mathfrak{sl}_d(\mathbb{R})$ since $R \in \mathrm{SO}_d(\mathbb{R})$ is an orthogonal matrix, and so $\mathrm{Tr}(R\bar{\mu}_0 R^T) = \mathrm{Tr}(R^T R\bar{\mu}_0) = \mathrm{Tr}(\bar{\mu}_0) = 0$. Note that in three dimensions we will need to specify another constraint on R in order to orient it uniquely with the brush direction, for example $R(e_y) = r$ for some vector r orthogonal to p.

Now we apply an impulse M_i to μ. Again we may either override the dynamics of μ_i entirely, clamp it to a maximum value, or linearly interpolate between μ_i and $\bar{\mu}_i$. The modification to Algorithm 2 is

$$\mu_{i+1} \leftarrow \mu_i + \dot{\mu}_i \delta s + M_i$$

for applying the impulse and

$$\mu_{i+1} \leftarrow (1-c)(\mu_i + \dot{\mu}_i \delta s) + c\bar{\mu}_i$$

for applying the control $\bar{\mu}_i$ with a blending parameter $c \in (0,1)$.

As in Section 4.3 we choose the latter interpolation method over the impulse method for a more intuitive control parameter.

Our fluid sketching application is extended to allow editing with 1-jet particles. We demonstrate the results by sketching with 1-jet particles, with Figure 4.5 showing a rotating flow, Figure 4.6 showing a scaling flow and Figure 4.7 showing a shearing flow. In the last two images, the orientation of $\bar{\mu}$ has been linked with the brush stroke direction to allow the user to align the vector field intuitively.

As with the 0-jet particle sketching, this technique is easily extended to three

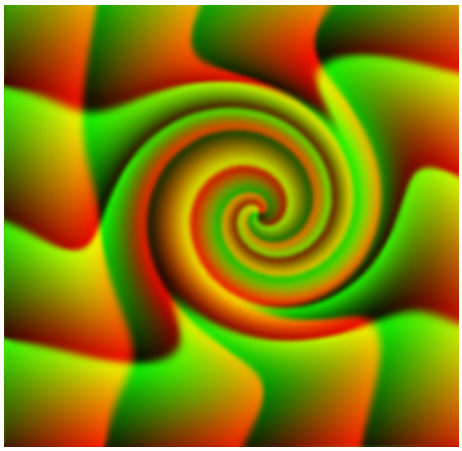


Figure 4.5: A constrained 1-jet *Spin* particle sketching a flow deforming a grid

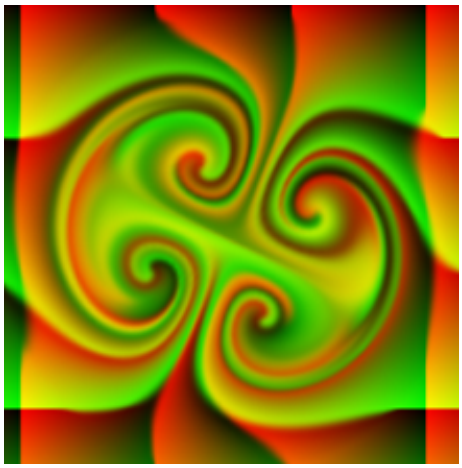


Figure 4.6: A constrained 1-jet *Scale* particle sketching a flow deforming a grid

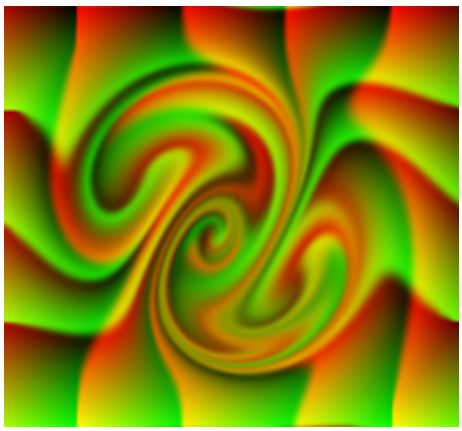


Figure 4.7: A constrained 1-jet *Shear* particle sketching a flow deforming a grid

dimensions. If we evaluate the velocity field with the same matrix kernel

$$u^i(x) = K^{ij}(q_\alpha, x)p_{\alpha i} - \mu_{\alpha j}{}^k \partial_k K^{ij}(q_\alpha, x) \qquad \text{(3.14 revisited)}$$

but now evaluated in three dimensions, we still generate a divergence-free velocity that is now parameterised additionally by μ. And as above we can choose a brush size σ, and constrain a 1-jet particle to the artist's 3D curve.

In $\mathbb{R}^3$ we can choose a basis for $\mathfrak{sl}_3(\mathbb{R})$ using matrices that represent scale, rotation and shear. An example of such a basis is shown here in Table 4.1.

Scale	$S_{xy} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$S_{xz} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$	$S_{yz} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$
Rotation	$R_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$	$R_y = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$R_z = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
Shear	$Sh_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$Sh_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$	$Sh_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

Table 4.1: A basis for the Lie Algebra $\mathfrak{sl}_3(\mathbb{R})$. Note we mark S_{yz} as red since we must omit one element to form a basis.

Note that the dimension of $\mathfrak{sl}(\mathbb{R}^3)$ is 8 since it corresponds to the sub lie algebra of trace free matrices in $\mathfrak{gl}(\mathbb{R}^3)$. Hence to strictly define a basis for $\mathfrak{sl}(\mathbb{R}^3)$ we must omit one element. We choose arbitrarily to remove S_{xy} as it can be generated from S_{xz} and S_{xy} since $S_{yz} := S_{xz} - S_{xy}$. For this reason we have marked it in red in Table 4.1. However all of the matrices in the table are useful to picture the space of vector fields spanned by the matrix kernels.

The vector fields generated in 3D by the 1-jet particles with μ taking the matrices defined above are shown in Figure 4.8. We use 1-jet particle flow in Chapter 5 to deform existing flows.

4.6 Keyframe Generation

We now show how we use the application we developed in Sections 4.3 and 4.5 for sketching fluid flows in order to generate keyframe edits of fluid simulations.

To demonstrate sketching of flows with jet particles with real simulation data, we extend our sketching application to allow editing of densities and velocities di-

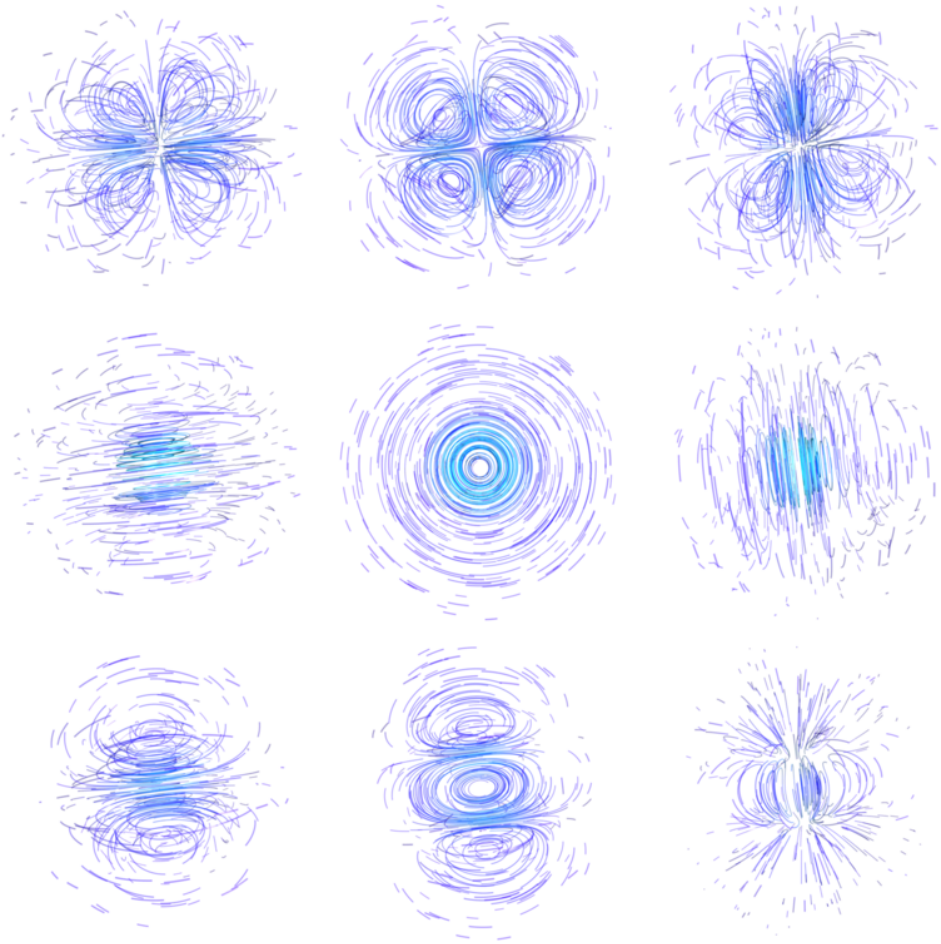


Figure 4.8: 1-Jet Particles in 3D. Top row Scale S_{xy}, S_{xz}, S_{yz}, middle row Rotation R_x, R_y, R_z and bottom row Shear Sh_1, Sh_2, Sh_3.

rectly using a jet particle that is constrained to the users mouse. We have integrated our application into the research simulation framework *mantaflow* [91] in order to allow sculpting of the current simulation state with jet particles. The deformation applies to the density and velocity fields stored at the frame being edited. It may also act on level sets and so is suitable for deforming water simulations.

The following video show a smoke simulation and a demonstration of sculpting edits interactively using our application.

Video 4.4 (2D smoke editing application)**.** A movie showing example edits applied to a 2D smoke simulation using 0-jet and 1-jet particles constrained to the mouse.

Figure 4.9 shows a single frame taken from the smoke simulation shown in Video 5.3. This frame is firstly sculpted with a 0-jet particle on the right and left

Figure 4.9: Smoke density and velocity field edited with 0-jet and 1-jet particle

side of the plume, before finally a 1-jet *spin* particle is used on the left side. The brush is displayed as a purple circle and its size defines the radius of the jet particle used. The size of the brush and the matrix $\bar{\mu}$ can be varied to sculpt the smoke as desired.

In Video 4.4, examples of creating edits to frames from a paused smoke simulation are shown. After each edit the simulation is restarted in its deformed state.

We believe that the ease with which this application enables us to sketch flows supports our Hypothesis 1 concerning the sketching of realistic flows. Moreover, once we have a jet particle simulation, we may define an edit to it by choosing a frame and then changing their momentum or simply moving the particles. Further particle constraints as discussed previously will give more sophisticated edits to the flow.

4.7 Dynamics of Jet Particles

In the previous sections, we have used the equations of motion for jet particles in our constrained simulations for sketching with a single particle. We now seek to create more complex interactions by leveraging the interaction between particles.

By solving the equations of motion given by Equations (3.10), (3.11) and (3.12) at each frame, jet particle simulations create complex flows and interactions. However, the dynamics are known to be unstable as we show in the following videos.

Video 4.5 (0-Jet particle simulation)**.** A video showing the simulation of several 0-jet particles with random initial momenta. The dynamics show complex interactions followed by particles coalescing as they become too close. The arrows represent the particle momenta. See Figure 4.10 for a clip from this movie.

Video 4.6 (1-Jet particle simulation)**.** A video showing the simulation of several 1-jet particles with random initial momenta. The dynamics show complex interactions followed by particles coalescing as they become too close. See Figure 4.11 for a clip from this movie. The arrows represent the particle momenta and the colours of the particles represent the vorticity of the particles, with red and blue indicating positive and negative vorticity, found by taking the anti-symmetric part of μ.

See Videos 4.5 and 4.6 for a demonstration of their instabilities that arise after a few seconds of simulation time. We have already seen clips from these videos as Figures 3.2 and 3.4 from Chapter 3 to illustrate their velocity fields. Those clips were taken from shortly after the start of the simulation.

In Figures 4.10 and 4.11 however, we show similar clips from the videos a few seconds later, and we can see that the momenta are beginning to blow up for pairs of particles that have become close.

When the particles get too close together they may become attracted to one another. In the limit they collide together, whilst their individual momenta become arbitrarily large [36] and this process can be seen in the videos. Moreover when the particles are too close with respect to the kernel size σ, their solutions are no longer close to Euler's equations.

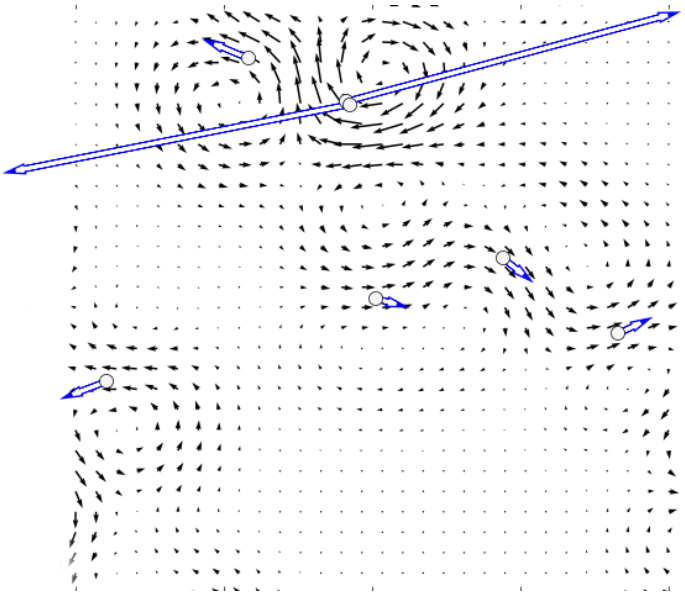


Figure 4.10: 0-Jet particle instabilities. A simulation of several 0-jet particles displaying instabilities.

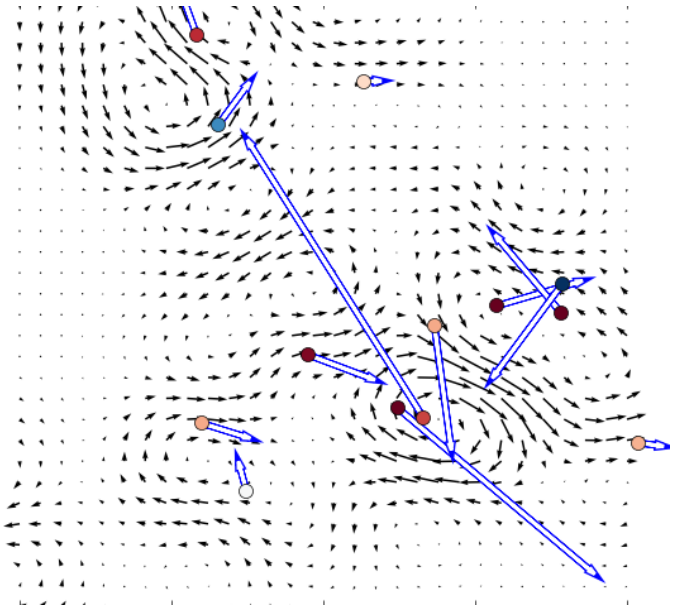


Figure 4.11: 1-Jet particle simulation. A simulation of several 1-jet particles displaying instabilities.

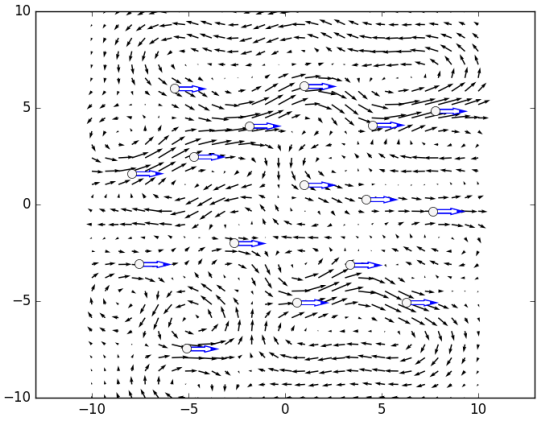


Figure 4.12: Initial conditions for the 0-jet particles

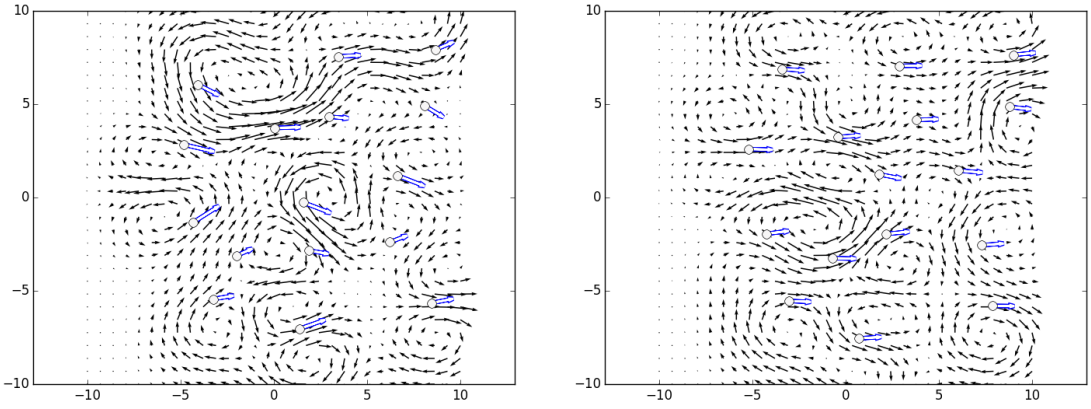


Figure 4.13: Unconstrained (left) vs constrained (right) 0-jet simulation after identical initial conditions. The constraint targets the particle momenta towards $(1,0)$.

However we show here that by adding constraints, we may keep the particle stable for longer and maintain the particle separation. This will keep the flow given by the particles closer to incompressible flow [36].

Video 4.7 (Jet particle constraints)**.** Simulation of 0-jet particles without constraints, and then with constraints on the momenta. The constraints blend the p towards the constant momenta of $p_0 = (1,0)$ in the x-direction, with a blend factor of $b = 0.1$.

Two frames from the associated Video 4.7 are displayed in Figure 4.13. They begin with the same initial conditions, shown in Figure 4.12. The images display

two 0-jet simulations at a fixed time interval following identical these conditions.

All the particles start with constant momentum $p_\alpha = (1,0)$, but the simulation on the right has a constraint which blends in 0.1 of the constant momentum $(1,0)$ per time step as described in Section 4.3. This blending represents the constraint of the simulated particles to a reference particle, using the momenta implied by the brush's motion.

The images show the progression of the simulation after a short interval, and we conclude that the constrained simulation shows less variation in the particles momenta and spacing because of our constraint.

Hence we can design flows with multiple particles in the following way. Instead of initialising one particle at the brush stroke position $\bar{q}_0$, we create N particles q_α scattered evenly within its radius. As before we solve the equations of motion for the particles and add the impulses from the brush $\bar{q}$ as a constraint. This time however, the interactions between the particles due to the Hamiltonian come into play as the velocity field of one particle affects another.

Multiple particles can also be used in three dimensions to create simulations for fluid generation. Figure 4.14 shows 30 interacting 0-jet particles with random momenta interacting in 3D, rendered with a trail to give an indication of their velocity. The simulation was created in Houdini, with the velocity fields evaluated as described in Appendix A.3.5.

This demonstrates that we can generate complex flows from a sparse representation of the fluid as jet particles. Further we can evaluate the velocity at arbitrary resolutions, and the velocity field will always be divergence-free.

4.8 Conclusions and Future Work

It is an open question as to whether this system is completely sufficient for artists. However, we have demonstrated a method to sketch fluid-like fluids in real-time at arbitrary resolutions, independent of the number of particles, and without requiring a simulation. Further, the velocity fields produced by Equation (3.5) is divergence-free so will guarantee incompressible flow. Hence the flows designed with this

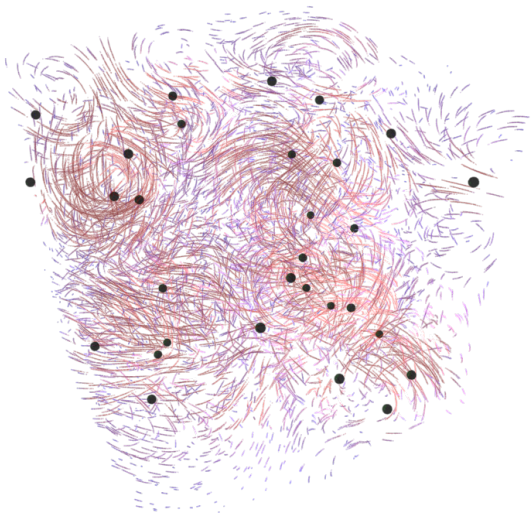


Figure 4.14: The velocity field from a simulation of 30 0-jet particles interacting in 3D.

method can provide an input to a fluid simulation engine at a higher resolution or can be combined with many existing techniques for adding turbulent detail [41].

Despite the fact that the jet particles become unstable as particles begin to collide and stick, efforts to keep the particles well placed may be successful in creating real-time simulators. Adding separation forces, or periodically reseeding the particles using techniques from Chapter 6.1 may allow the simulation to continue without instabilities. Moving the particles to the center of their voronoi diagrams [92] every few time-steps may be enough to keep the particle simulation stable and hence keep the solutions close to Euler-α.

Compound brushes may also be defined to create more interesting flows. We can constrain several jet particles to one brush stroke with different sizes and momenta, for example we can combine a large 0-jet particle surrounded by smaller 1-jet satellite particles. This is similar to a method for combining several particles to create a *module* as defined by Gris, Durrleman, and Trouvé [93] for building deformations within the LDDMM framework.

Finally, if we sketch a flow with a multiple series of brush strokes, it would useful to represent this composition of diffeomorphisms into a single flow given by multiple particles in series instead of in parallel.

The next chapter goes on to find methods for editing existing flows, and we

will use the fluid sketching approach developed in this chapter to augment existing flows in a natural way.

Chapter 5

Editing Fluids with Jet Particles

Editing fluid simulations in a way that keeps the fluid-like appearance of the flow is a hard problem to solve. We intend to show that jet particles bring a natural way to edit flows, and we will demonstrate this by means of some concrete examples. To this end, we take the techniques from the previous chapter for designing fluid flows with jet particles, and couple them with existing fluid simulations to provide a fluid editing system in a production environment.

In Section 5.1 we describe how to modify an existing flow with a jet particle flow acting *on the left* as a post deformation, and we give simple examples in two and three dimensions. We compare the technique with the skeletal subspace deformation method commonly used in computer graphics.

Section 5.2 describes a simple method for coupling jet particle flow with an existing flow through a rigid transformation. Section 5.3 describes a natural way to advect a flow given by jet particles using an existing simulation. And in Section 5.4 we generalise this idea by providing a method for pushing forward the jet particles with a flow.

In Section 5.5 we show a way of treating the edit as a deformation keyframe, which only affects the fluid cache locally in time. Finally in Section 5.6 we show how this technique is used in a production environment to create variations of smoke simulations.

5.1 Editing Existing Flows

In this section we take a single jet particle and sketch a fluid flow as in Chapter 4. We will now couple this flow with an off-line, or *cached* simulation to create a simple edited flow.

The cached simulation may include grids storing densities and scalar fields such as temperature and heat, as well as vector fields such as velocities. The most natural way to achieve this coupling is to apply the sketched flow as a deformation acting on the fluid's domain. In Sections 5.2 to 5.5 we also require that the cached input simulation should define a flow field. We will assume that this is provided as velocity vectors for each frame, which we will use for advecting and coupling our edits with the cached flow.

The application of the edit to the cached simulation is achieved by composing the two flows. In particular the jet particle flow is composed on the left with the original simulation: we flow with the cache, and then apply our edit as a secondary flow. So for each frame of our cached simulation, we apply a deformation given by the jet particle edit flow. At each subsequent frame of the cached simulation, we apply the whole edit again whilst adding a new frame from the jet particle simulation, so that the effect of the edit will increase visibly over time. In this way we couple the two times together, since the edit time is linked with the simulation time.

Any velocity field $u(x,t)$ will create a flow over time given by the function $\varphi(x,t)$. The relationship between the flow and the velocity field is

$$u(\varphi(x,t),t) = \frac{\partial \varphi}{\partial t}(x,t)$$

and since in our case $u(\cdot,t)$ is a divergence-free vector field we have that $\varphi(\cdot,t) \in \mathrm{Diff}_{vol}(\mathbb{R}^d)$ is in the space of volume preserving diffeomorphisms. So we can think of our cached fluid simulation as defining a flow $\varphi(x,t)$.

In the same way, the velocities from a jet particle flow also create a flow which we call $\psi(x,t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$. This is another incompressible flow in $\mathrm{Diff}_{vol}(\mathbb{R}^d)$.

We now take the composition of the two flows $\varphi'(x,t)$ given by

$$\varphi'(x,t) := \psi \circ \varphi(x,t) = \psi(\varphi(x,t),t)$$

to give our new edited flow, which is also volume preserving by construcution.

5.1.1 Comparison with Skinning

We propose to use jet particle flow to edit existing fluid simulations. To our knowledge, jet particles have not been used to edit fluid simulations to date. However it is our opinion that they give natural deformations for the purpose, as their flow is strongly linked to Euler's equations, as outlined in Section 3.7.

The most common technique in today's animation pipelines for deforming geometry is skeleton-subspace deformation, also know as *skinning*. The skinning algorithm is widely used and has not been published in the literature, but see Lewis, Cordner, and Fong [94] for a review of skinning techniques and references therein. The skinning method describes a deformation based on the transformation of a number of matrices often called *bones* and scalar *blend weights* that are assigned to each point. Each matrix typically contains a translation, rotation and scaling element only.

The videos that follow show a comparison of a simple translational and rotational flow given by a single jet particle (marked by a circle) with the same corresponding deformations achieved with skinning by a bone (marked with a square).

Video 5.1 (Comparison of 0-Jet flow with skinning by translation)**.** This video shows the baseline deformation given by skinning using one translating bone, and compares it with the corresponding flow given by a 0-jet particle with the same momentum as the bone. A clip from the video is shown in Figure 5.1.

Video 5.2 (Comparison of 1-jet flow with skinning by rotation)**.** This video shows the baseline deformation given by skinning using one rotating bone, and compares it with the corresponding flow given by a 1-jet particle with a μ representing the rotation. A clip from the video is shown in Figure 5.2.

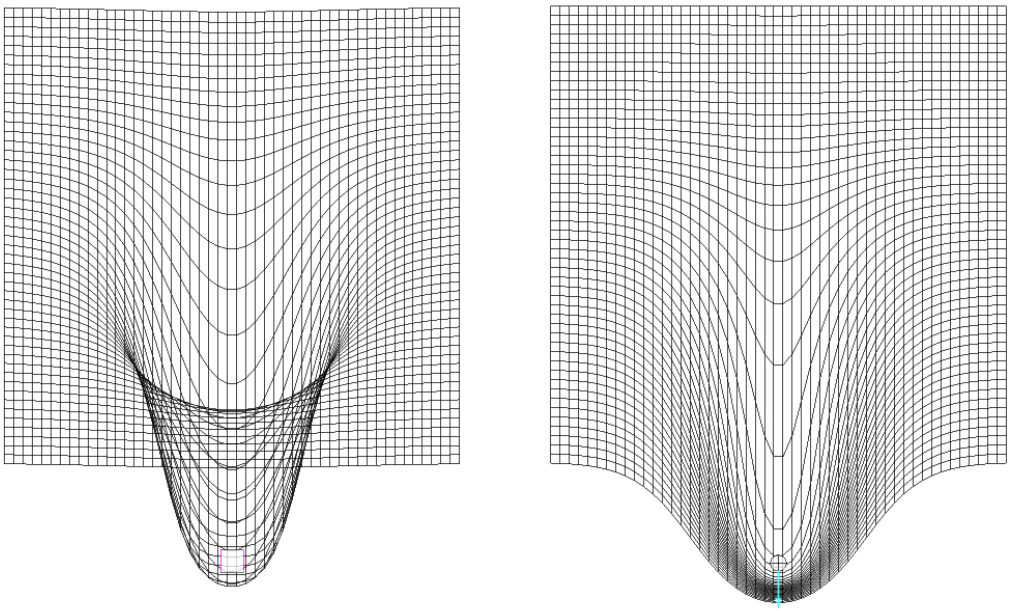


Figure 5.1: Comparison of a 0-jet flow with skinning by translation. The skinning deformation is on the left and the 0-jet particle flow is on the right.

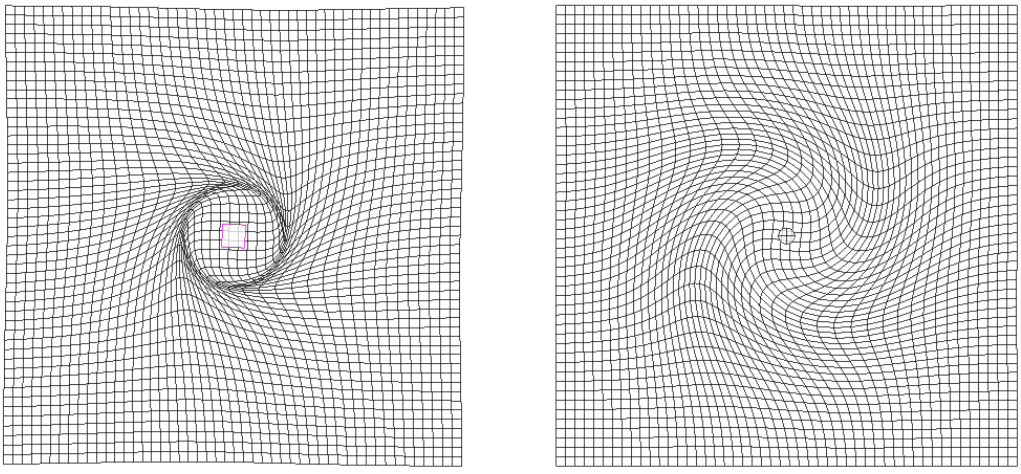


Figure 5.2: Comparison of a 1-jet flow with skinning by rotation. The skinning deformation is on the left and the 1-jet particle flow is on the right.

We can see from Videos 5.1 and 5.2 how the translating or rotating bone will generate a similar flow to the jet particle flow initially.

However it is clear that the skinning deformation is not volume preserving, and the mapping also fails to be injective after a short time, whereas the jet particle flow stays injective and is incompressible. Hence we believe that jet particle flow represents a natural solution for creating deformations that are fluid-like, and is a good candidate for editing existing flows compared to the existing skinning technique.

5.1.2 Deformation Metric

We can quantify the edit by calculating a length given by the flow using the reduced metric on the jet particles

$$length = \int H(q,Q,p,P)^{1/2} dt \tag{5.1}$$

where the Hamiltonian for the 0-jet particles is given by Equation (3.4) and for 1-jet particles by Equation (3.7).

So for example, in the case that we are only considering one 0-jet particle for the fluid sketching, we can define the size of the edit as

$$length = \int \left(p^T K(0) p\right)^{1/2} dt$$

which in this case is just $\int \left(p^T p\right)^{1/2} dt = \int |p| dt$, which follows since $K(0) = \mathbb{I}_d$ in the case where we have only one particle.

This can be used to quantify the size of the deformation. As the brush stroke is being recorded we can now indicate the size of the edit for the user by displaying this length.

We now show examples of deforming some fluid flows in two and three dimensions using this approach.

5.1.3 Deforming 2D Smoke Simulations

Here we show the result of deforming a frame from a 2D smoke simulation with the Hamiltonian flow given by a single jet particle. In each example the upper image is

the original and the lower image is the deformed version.

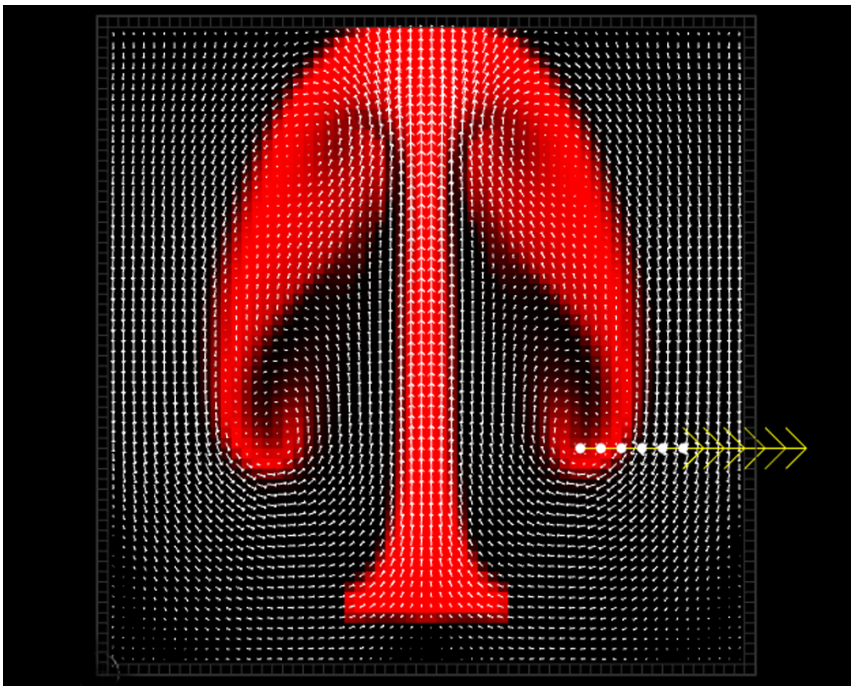
Video 5.3 (Original 2D Smoke Simulation)**.** This video shows a 2D smoke simulation generated from a buoyant source at the bottom of the grid. A clip from the video is shown in Figure 5.4a.

Video 5.4 (Deformation of 2D Smoke with Static 0-jet particle)**.** This video shows the previous 2D smoke simulation which has now been deformed with an edit given by the flow of a 0-jet particle. A clip from the video is shown in Figure 5.3b.

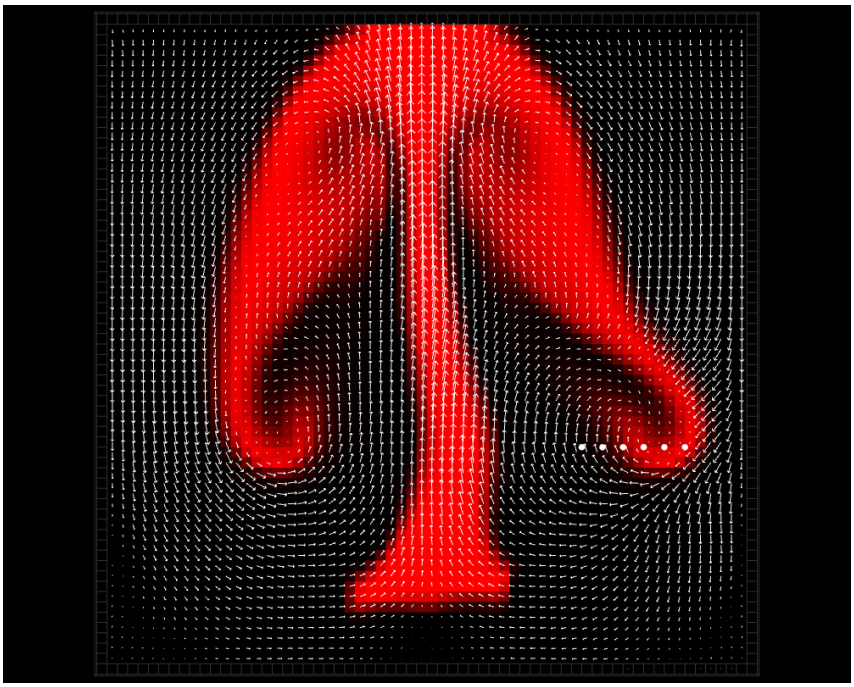
The first example, shown in Video 5.4 shows a deformation that moves a visually significant feature of the smoke plume to the right, and our second example rotates the same feature.

Figure 5.3 shows a frame from the original 2D smoke simulation and the result of deforming it with the flow given by a 0-jet particle propagating along the x-axis. Figure 5.4 shows the second example of a frame from a 2D smoke simulation and the result of deforming it with the flow given by a 1-jet *Spin* particle with zero momentum p.

These figures have been generated in a version of *mantaflow*[91] that we have modified to use the forwards deformation techniqued described by Algorithm 7 in Section 7.1. This includes the push-forward of the original velocity by the deformation.

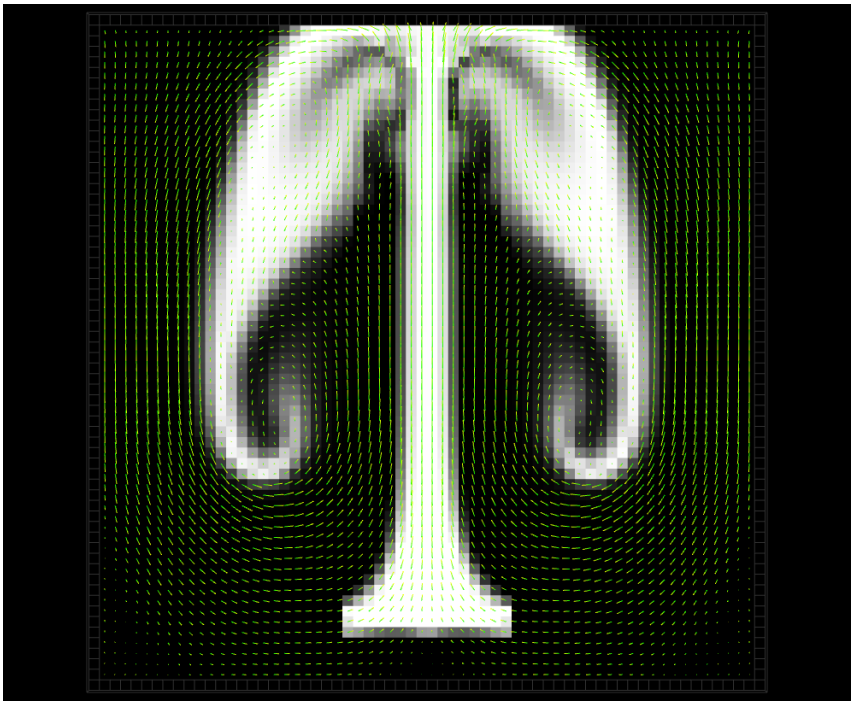


(a) Original smoke density, with the jet particle motion edit path indicated with dots, and its momentum indicated with arrows



(b) Deformed smoke density with the 0-jet particle flow, with the jet particle edit path indicated with dots

Figure 5.3: Editing of the 2D smoke simulation with a 0-jet particle flow



(a) Original smoke density from the simulation

(b) Deformed smoke density with the 1-jet particle flow, given by a jet particle at the position marked by the dot

Figure 5.4: Editing of the 2D smoke simulation with a 1-jet particle flow

5.1.4 Deforming 3D Smoke Simulations

We show a 3D smoke simulation in Video 5.5, and edited versions where it is deformed by the flow given by a 0-jet and a 1-jet particle simulation in Videos 5.6 and 5.7.

Video 5.5 (Original 3D smoke simulation)**.** A video of a 3D smoke simulation simulated in Houdini. A clip from the video is shown in Figure 5.5a.

Video 5.6 (3D smoke deformed with 0-jet particle)**.** The 3D smoke simulation with the 0-jet flow applied as an edit. A clip from the video is shown in Figure 5.5b.

Video 5.7 (3D smoke deformed with 1-jet particle)**.** The 3D smoke simulation with the 1-jet flow applied as an edit. A clip from the video is shown in Figure 5.6b.

These videos illustrate how the jet particle flow can be coupled with the original flow. The deformations are implemented using the technique described in Section 7.3 where the edit flow is calculated on a sparse grid and then interpolated into the rest of the grid.

Although we have demonstrated the editing of a fluid simulation, it is clear from these videos that this basic application of the edit flow from the jet particles to the original flow does not give physically plausible results. Recognising this fact, we propose a method to track the edit flow with the off-line simulation in order to link the fluid flows together in a physically significant way. We expand on this in the next section.

(a) Original smoke density

(b) Deformed smoke density

Figure 5.5: Deformation of a 3D smoke simulation with a 0-jet particle flow

(a) Original smoke density

(b) Deformed smoke density

Figure 5.6: Deformation of a 3D smoke simulation with 1-jet particle flow

5.2 Tracking Jet Particles

In Section 5.1 we created a new flow by composing the cached flow with one that we designed using jet particles. However the results do not seem physically realistic because there is no coupling between the two flows.

We can couple the original fluid flow with the sketched flow in a simple way, by moving the coordinate frame of the sketched flow with the fluid. To this end, we take the initial position of the sketched particle and advect it with the fluid.

The action of the sketched flow can be faded out over time, by retracting the deformation over some simulation time interval. Alternatively the size of the particle kernel can be reduced to zero during simulation time although this leads to sharper features, so we favour the first approach which we describe below.

The following method will simply pick up the deformation given by the jet particles and *pin it* to the fluid:

- Choose an edit time for the deformation and call this time $t = 0$
- Sketch a flow with jet particles $J_\alpha(s) = (q_\alpha(s), p_\alpha(s), \mu_\alpha(s))$ at time t
- Take the initial position of the jet particles $q_\alpha(0)$ and advect it with flow $\varphi(x,t)$ to give the tracked position $\varphi_t(q_\alpha(0))$ where $\varphi_t(x) := \varphi(x,t)$
- Create a new 1-parameter family of jet particle flows, where s is the edit time and t is the new parameter given by the flow:

$$L_\alpha(s,t) = (q_\alpha(s) + dq_\alpha(t), p_\alpha(s), \mu_\alpha(s)) \tag{5.2}$$

  where

$$dq_\alpha(t) = \varphi_t(q_\alpha(0)) - q_\alpha(0)$$

  is the displacement vector, constant for each jet particle α at time t.
- Now for each time t we have a different jet particle edit which is tracked to the fluid
- Write $\psi_t(x,s)$ to be the flow induced by the jet particles $L_\alpha(s,t)$ at time t
- We now combine the flows by composition to give the new flow $\psi_s \circ \varphi_t :=$

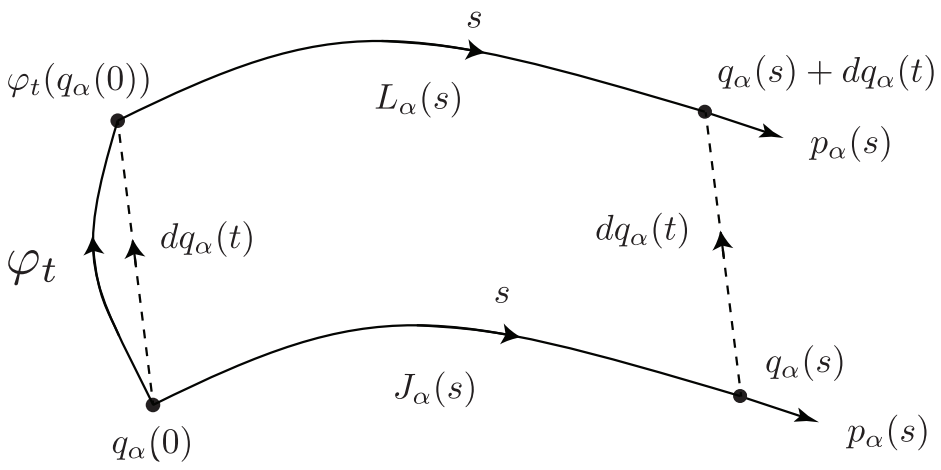


Figure 5.7: The new jet particles $L_\alpha(s)$ are created by offsetting the jet particles $J_\alpha(s)$ with the displacement vector $dq_\alpha(t)$, given by the flow φ_t applied at $q_\alpha(0)$

$\psi(\varphi(x,t),s)$

- This is a 2-parameter family of flows: flow by φ for simulation time t then flow by ψ for edit time s.
- We can pick an edit by setting $s := t$ and consider the flow $\psi_t \circ \varphi_t$
- This will simply take the particles $J(s)$ tracked with the flow φ and incrementally apply their deformation over time t.

Figure 5.7 shows a jet particle flow J_α moving rigidly as its base point is carried with the flow. Note that the two curves $J_\alpha(s)$ and $L_\alpha(s)$ are deliberately drawn as parallel, with fixed offset $dq_\alpha(t)$. This is in order to indicate that the entire jet particle flow is tracking rigidly with the off-line fluid cache.

We can choose a regularised velocity field to advect the edit simulation frame to avoid picking up the high frequencies in the simulation velocity field. To do this we can use the smoothing kernel described in Holm, Nitsche, and Putkaradze [95], choosing a kernel size that is of the order brush size used for the edit. Alternatively we can simply create a low resolution version of the flow using resampling in order to advect the simulation frame, as we have done here.

The following Videos 5.8, 5.9 and 5.10 show examples of coupling edit flows defined by jet particles with the 3D smoke simulation shown in Video 5.5.

Video 5.8 (3D smoke deformed with an advected 0-jet particle)**.** The 3D smoke simulation with the 0-jet flow applied as an edit, which is tracked with the off-line simulation to couple it with the flow. A clip from the video is shown in Figure 5.8b.

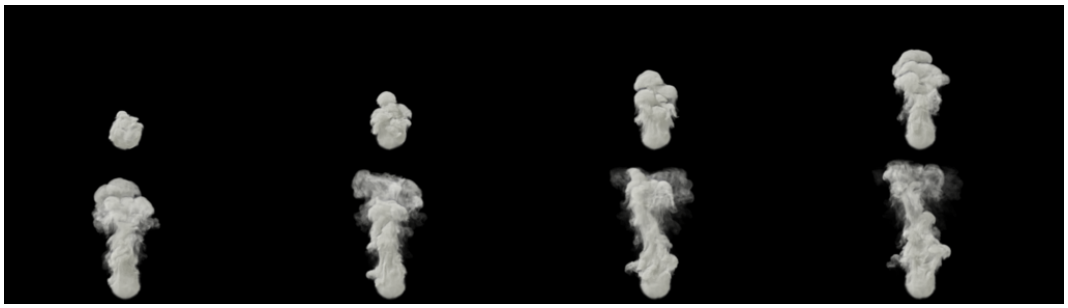
Video 5.9 (3D smoke deformed with an advected 1-jet particle)**.** The 3D smoke simulation with the 1-jet flow applied as an edit, which is tracked with the off-line simulation to couple it with the flow. A clip from the video is shown in Figure 5.8c.

Video 5.10 (3D smoke deformed with multiple advected 1-jet particles)**.** The 3D smoke simulation with the multiple 1-jet flow applied as an edit, which is tracked with the off-line simulation to couple it with the flow. A clip from the video is shown in Figure 5.8d.

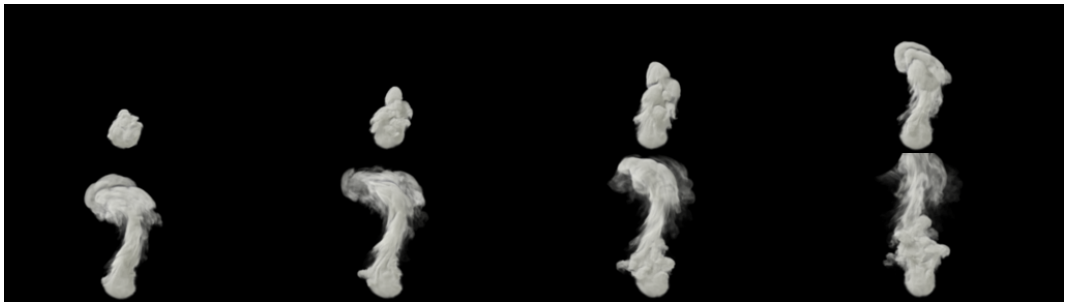
See Figure 5.8 for clips from these videos showing examples of coupling the original smoke simulation with advected edits given by jet particle flow.

These figures show the original simulation in Figure 5.8a, the simulation coupled with a 0-jet edit flow in Figure 5.8b, the simulation coupled with a rotating 1-jet edit flow in Figure 5.8c, and finally the simulation coupled with 64 random rotating 1-jet particles in Figure 5.8d in order to model the effect of turbulence.

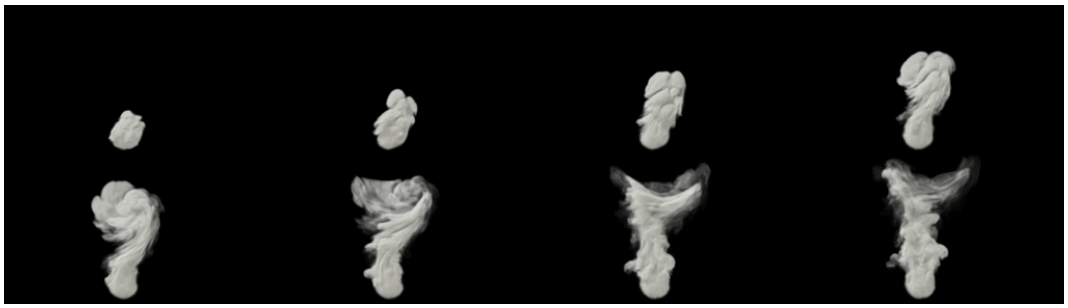
We believe that the coupling of a realistic sketched flow with an existing off-line cache in this way supports our Hypothesis 2 concerning the realistic editing of fluid flows.



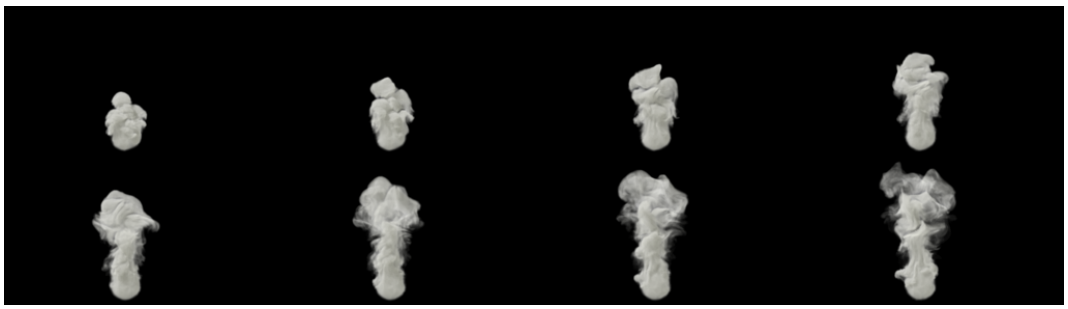
(a) Original simulation



(b) Simulation deformed with an advected 0-jet particle



(c) Simulation deformed with an advected 1-jet particle



(d) Simulation deformed with 64 advected 1-jet particles

Figure 5.8: Deformation of a 3D smoke simulation using advected jet particle flow, showing 8 selected frames in increasing time in left-right minor and top-bottom major order.

5.3 Advecting Jet Particles

In Section 5.2 we achieved our aim of coupling our jet particle flow with the cached flow, by moving its coordinate frame rigidly with the flow. Now, in order to have

a deeper coupling between the jet particles and the flow, we will advect the jet particles for all edit times s.

The idea here is to start with an artist designed deformation from Chapter 4 and extrapolate it into the flow using advection.

When we advect the jet particle path with the input fluid simulation, the brush stroke will become stretched and noisy due to turbulence. Neither of these properties is useful when designing a local deformation. We choose to minimise this effect by using a regularised version of the flow: we use a low resolution version of the velocities to remove frequencies smaller than the order of the kernel size used for the edit.

In our implementation, we resample the space curve given by the artist's brush stroke X at equal intervals of edit time δr to give m points Y_i. We now advect the points forwards and backwards into the flow to give m new points $Y_i(t)$ for each time $t \in (T-\varepsilon, T+\varepsilon)$. These points $Y_i(t)$ are hence functions of simulation cache time t.

Now that we have a brush stroke defined for each time t, we can find an incompressible flow for each time t using the flow sketching method described in Chapter 4.

The points $Y_i(t)$ are sufficient input for a constrained 0-jet simulation since the positions $Y_i(t)$ provide a reference particle path $(\bar{q}(s,t), \bar{p}(s,t)) := (Y(s,t), \frac{dY(s,t)}{ds})$ as before. Here the curve $Y(s,t)$ interpolates the points $Y_i(t)$ for a time t held fixed, and $s \in (0, s_{\max})$ as before.

In the case of a 1-jet constrained simulation we need to also provide a reference $\bar{\mu}(s,t)$ by the same method. Lastly if we are simulating multiple jet particles we advect just their start positions with the flow and proceed as before to integrate the vector fields into a flow but this time at time $t \neq T$.

5.4 Push-forward of Jet Particles

The method from the Section 5.3 can be generalised in a similar way to Section 5.2 by modifying Equation (5.2) with the advection of the jet particles $J(s)$ flow φ.

If we record the jet particles' initial conditions, we can push them forward with the flow before simulating the particle at the next frame.

We can push forward the momentum p with the deformation gradient Q to give Qp. Note that μ_α has one-up index and one-down index, i.e. ${\mu_{\alpha i}}^j$, so it transforms like $\tilde{\mu}_i{}^j = \mu_k{}^\ell \partial_i \phi^k(q) \partial_\ell \phi^j(q)$. Hence the push forward of μ by the flow is $Q\mu Q^\mathsf{T}$.

So now we create a new jet particle flow $L(s)$ defined by

$$\begin{aligned} L_\alpha(s,t) = &\quad (\varphi(q_\alpha(s),t),\, \varphi_*(q_\alpha(s),t) \circ p_\alpha(s),\, \varphi_*(q_\alpha(s),t)\mu_\alpha(s)\varphi_*(q_\alpha(s),t)^\mathsf{T}) \\ = &\quad (\varphi_t(q_\alpha(s)),\, \varphi_{t*}(q_\alpha(s)) \circ p_\alpha(s),\, \varphi_{t*}(q_\alpha(s))\mu_\alpha(s)\varphi_{t*}(q_\alpha(s))^\mathsf{T}) \end{aligned}$$

where we have pushed forward the entire jet particle flow J_α by φ_t. Now the whole jet particle path is stretched by the flow.

In this way the momentum p_α is calculated directly with the push-forward rather than using the approximation given the next particle position $p_{\alpha+1}$. Also the transformation of μ_α now takes the flow into account directly. In fact this is the continuous formulation of the advection problem from Section 5.3 where we used a discretised approach.

Figure 5.9 illustrates a jet particle path J_α deforming with the flow φ_t. Note that the two curves $J_\alpha(s)$ and $L_\alpha(s)$ are no longer parallel due to the flow φ_t being applied at each point.

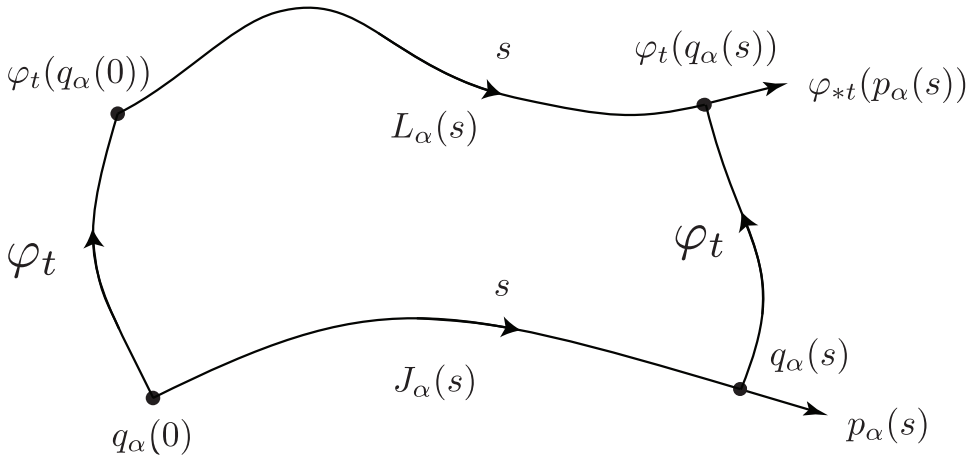


Figure 5.9: The new jet particle flow $L_\alpha(s)$ is calculated by deforming the jet particle flow $J_\alpha(s)$ with the cached flow φ_t

Video 5.11 (Deforming an existing smoke simulation with a coupled 1-jet particle flow)**.** We deform an existing smoke simulation with 32 turbulent 1-jet particles that are advected with the flow. The original flow is on the right, and the deformed flow is on the left. An image from the movie is shown in Figure 5.10.

Figure 5.10 shows a frame from Video 5.11 demonstrating an example from production where an off-line smoke simulation has been deformed by 32 1-jet particles, seeded with random momenta and advected with the flow. This illustrates how we can generate turbulence-like effects by the composition of an existing flow with a jet particle flow.

5.5 Deformation Keyframes

In Sections 5.3 and 5.4 we advected jet particles that describe a deformation into an existing flow φ.

In order to localise our modification to the flow in time, we suggest a method for coupling the retraction of the jet particle deformation. To achieve this, after a small window of time the edit is removed and we return to the original undeformed fluid simulation.

Specifically if the artist designs a flow at time T, we would like to extrapolate the deformation so that it acts over the interval $(T-\varepsilon_1, T+\varepsilon_2)$ for some small time

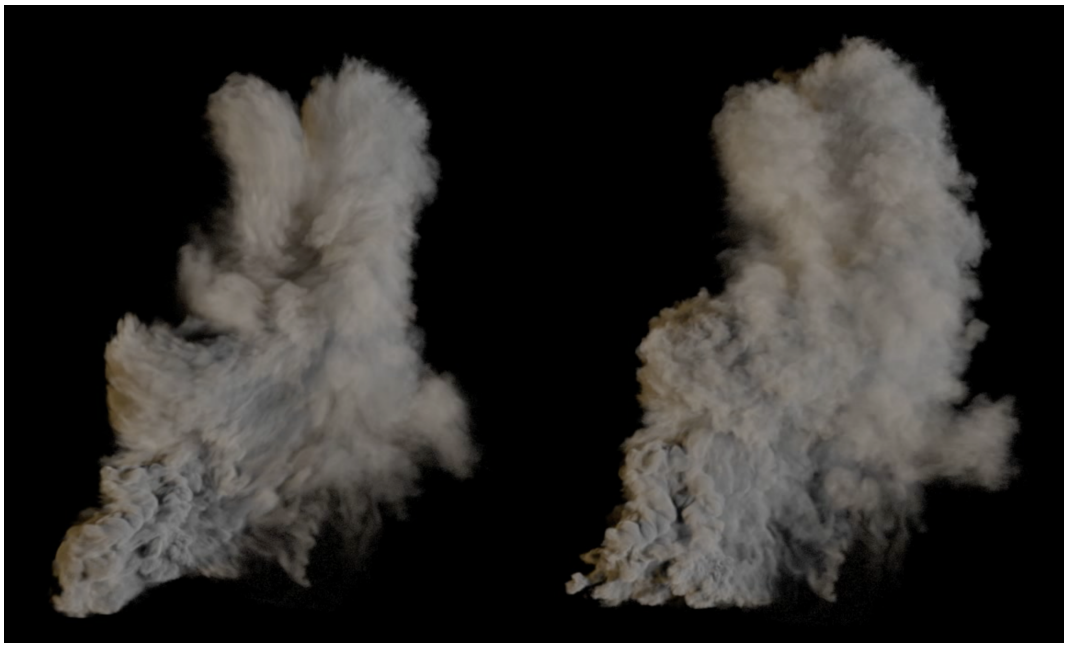


Figure 5.10: Deformation of an existing flow with turbulent 1-jet particles. The original simulation density at a frame is shown on the right, with the resulting deformed density displayed on the left.

deltas $\varepsilon_1, \varepsilon_2 > 0$. We will assume for simplicity that $\varepsilon = \varepsilon_1 = \varepsilon_2$ although this need not be the case.

We seek a deformation that retracts while it is extrapolated into the flow, so that at times $T - \varepsilon$ and $T + \varepsilon$ the original undeformed flow is restored.

In order to choose a way of blending in the deformation over time, we must first specify a smooth curve $\beta : [-\varepsilon, \varepsilon] \to [0,1]$ such that $\beta(-\varepsilon) = \beta(\varepsilon) = 0$, $\beta(0) = 1$ and $\dot{\beta}(-\varepsilon) = \dot{\beta}(0) = \dot{\beta}(\varepsilon) = 0$. We use a quadratic B-spline that satisfies the above properties, shown in Figure 5.11.

We use this function to scale down the distance travelled by the jet particle along the curve $Y(s,t)$. In this way, the deformation is retracted over time ε.

Hence the retracting jet particle reference path at time t is given by the curve

$$Y(\beta(t-T)s, t)$$

at time t. In this way the deformation retracts as the curve is advected back and forward in time from T.

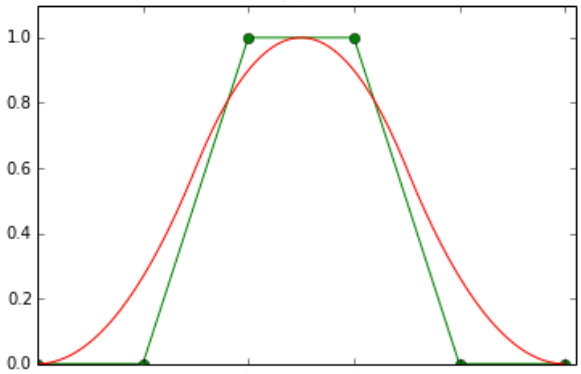


Figure 5.11: The spline function β used for retracting the deformation.

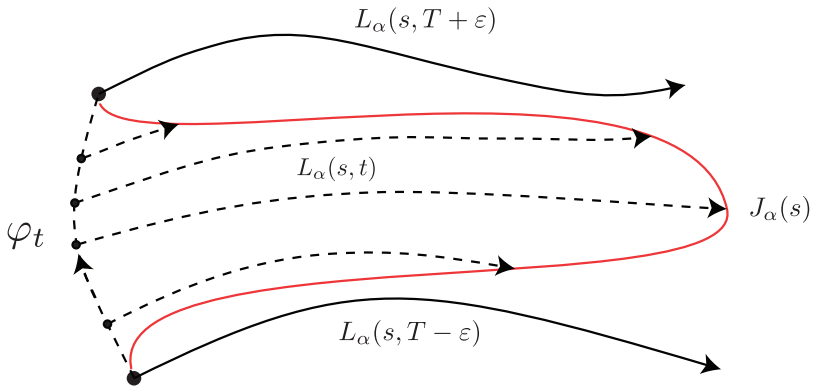


Figure 5.12: Retracting the edit flow $\psi(s)$ around time T, by applying the curve β to the edit time s in the jet particle flow $L_\alpha(s,t)$.

We can formulate this edited flow in the language of Section 5.4 by defining our new flow as

$$\psi_{\beta(t)} \circ \varphi_t = \psi(\varphi(x,t), \beta(t))$$

where the function $\beta(t)$ applies the jet particle flow $L(s,t)$ only in a neighbourhood of $t = T$.

See Figure 5.12 for an example of four different interpolated flows shown by dotted lines, each generated in the neighbourhood of the edit at time T.

In order to demonstrate the retraction of the jet particle flow, we take the orig-

inal 2D smoke simulation from Video 5.3. By applying the edits shown in Figures 5.3 and 5.4 and retracting them into the flow, we create the following videos:

Video 5.12 (2D smoke 0-jet deformation keyframe edit)**.** The 2D smoke simulation with the 0-jet flow applied as an edit, which is advected and retracted with the off-line simulation to define a localised keyframe edit of the flow. Some representative frames from the video are shown in Figure 5.13.

Video 5.13 (2D smoke 1-jet deformation keyframe edit)**.** The 2D smoke simulation with the 1-jet flow applied as an edit, which is advected and retracted with the off-line simulation to define a localised keyframe edit of the flow. Some representative frames from the video are shown in Figure 5.14.

We extend and then retract the 0-jet and 1-jet edit flows into the off-line smoke simulation, and show the results in Videos 5.12 and 5.13. See a sequential selection of stills from these videos in Figures 5.13 and 5.14.

In the next section we show how to create edited flows in 3D using a curve rig.

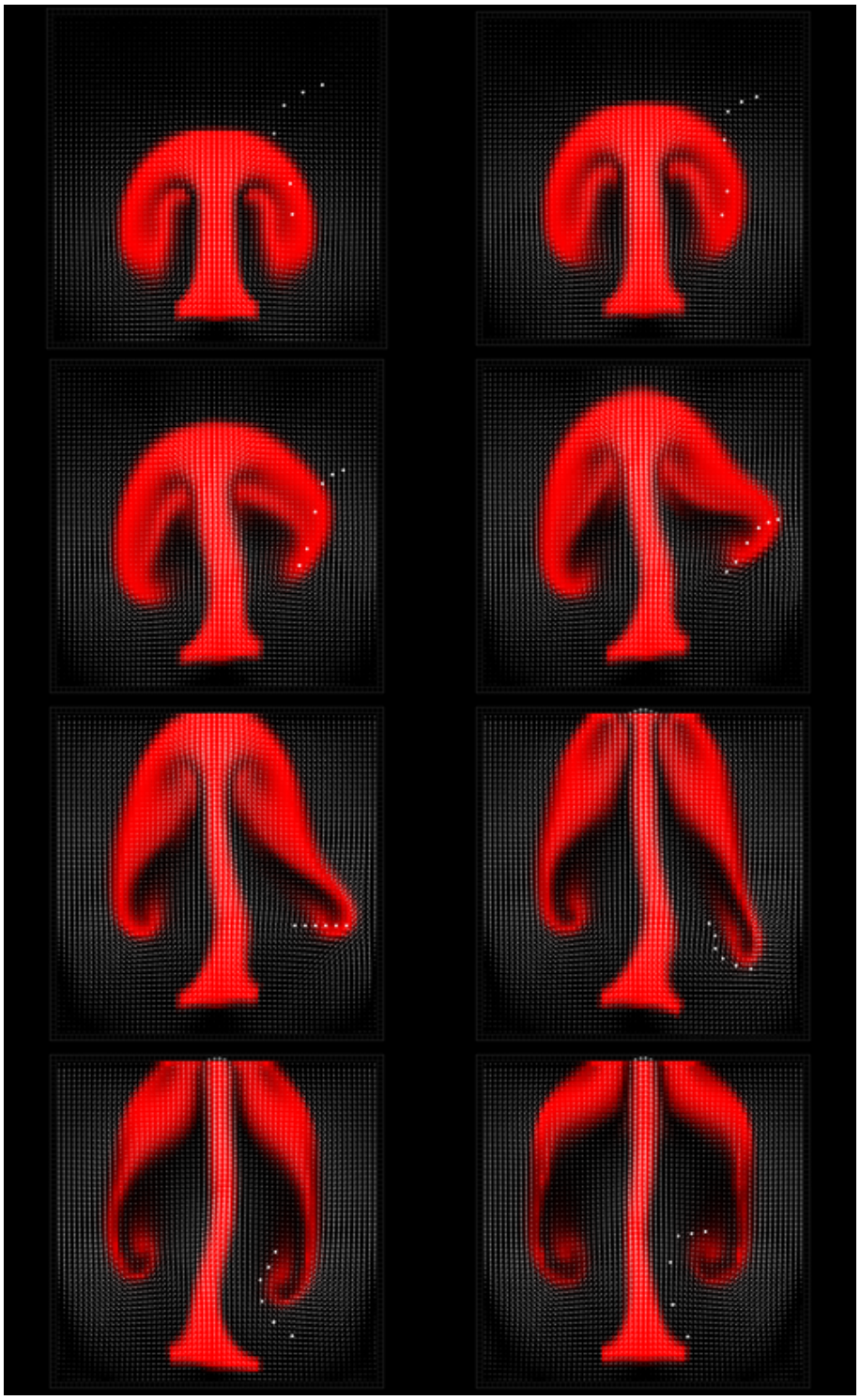


Figure 5.13: A 2D smoke simulation deformed by a retracted 0-jet edit flow. The advected jet particle flow path before retraction is indicated by dots. Frames are in left-right minor and top-bottom major order.

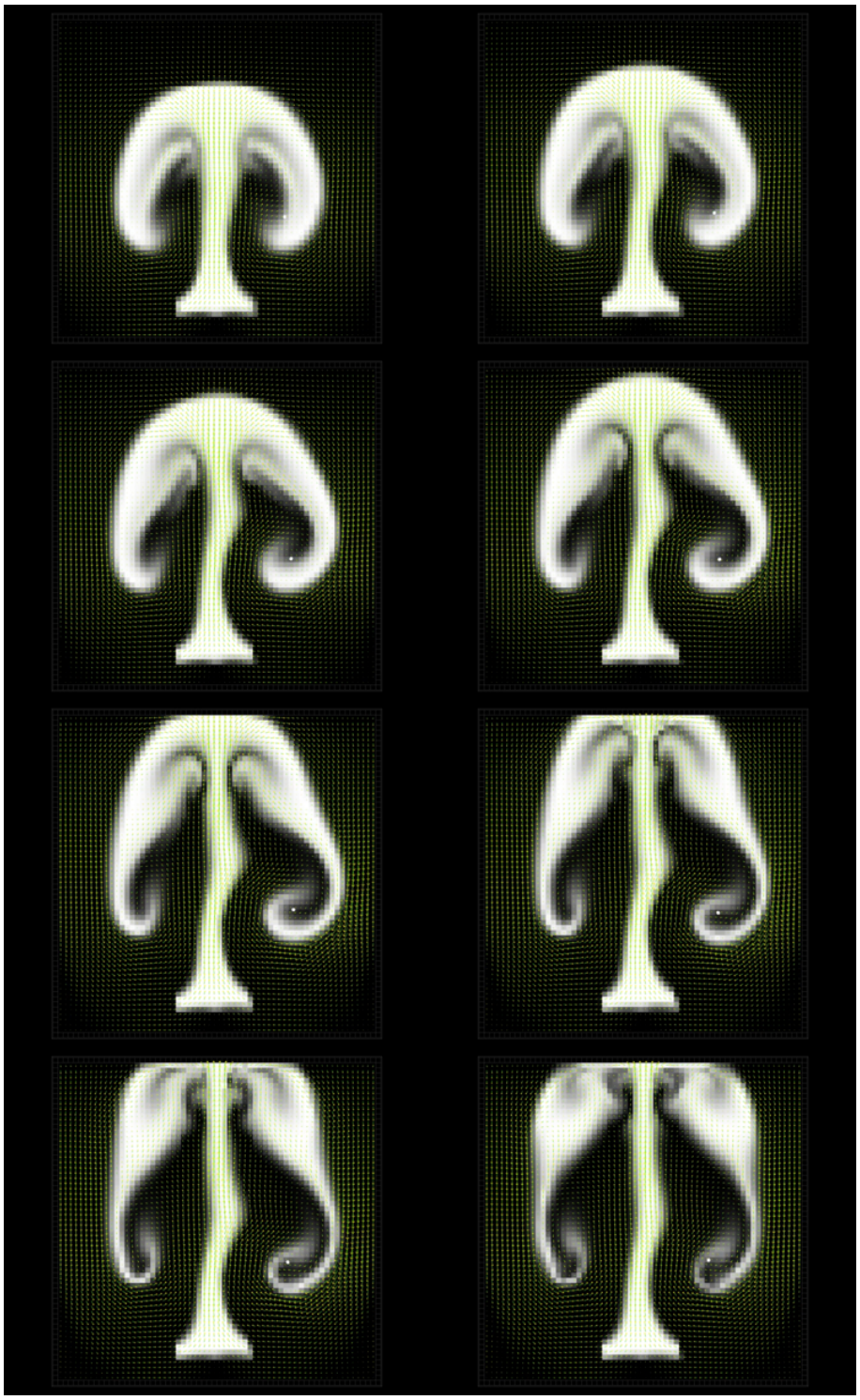


Figure 5.14: A 2D smoke simulation deformed by a retracted 1-jet edit flow. The advected jet particle flow path is indicated with a dot. Frames are in left-right minor and top-bottom major order.

5.6 Jet Particle Rigs in Maya

In order to be able to deform fluid caches easily for shot production, we designed a simple work flow in Autodesk Maya 2016 for designing flows with jet particles.

To prepare the scene, the artist simply creates a curve representing the path of a jet particle's motion. Using the Maya representation of a motion path, the start and end frame of the motion are configured easily. Additionally the μ parameter for the 1-jet particle can be set to be a combination of scale, rotate and shear as described in Section 4.5.

In order to preview the effect of the deformation at interactive rates, a low resolution proxy version of the fluid cache is calculated automatically after simulation. This is then deformed inside Maya as described in Section A.3.3. Finally the high resolution deformed smoke grids are calculated directly on the artists machine or sent to the render farm for remote processing.

In this section we show an example of a shot from a real world production, from effects work on Independence Day 2 at Cinesite Studios. An image rendered from the simulation is show in Figure 5.15. After the shot had been included in a viewing session, feedback from the client was given that the smoke stack should lean over as if being blown by a wind.

To address this, the simulation was run again with a wind force included. These simulations were taking 9 hours to complete on an Intel i7, and creating density grids of size $500 \times 1000 \times 500$ voxels.

A realistic alternative to re-simulation is to apply a post deformation to the fluid cache. By composing the flow with a simple deformation derived from 0-jet particle flow, we are able to simulate the effect of the wind and generate a suitable result for the shot. An advantage to this technique is that the results can be reviewed and tweaked in low resolution efficiently, before calculating the final high quality version.

See Figure 5.16 for a frame from the sequence with the deformation applied. The deformation was designed such that the smoke, which was previously rising almost vertically, would appear to be driven by a wind blowing from screen left.

Figure 5.15: Smoke densities from a simulated smoke stack used in a shot in production.

Figure 5.16: The deformed smoke densities found by applying the jet particle flow.

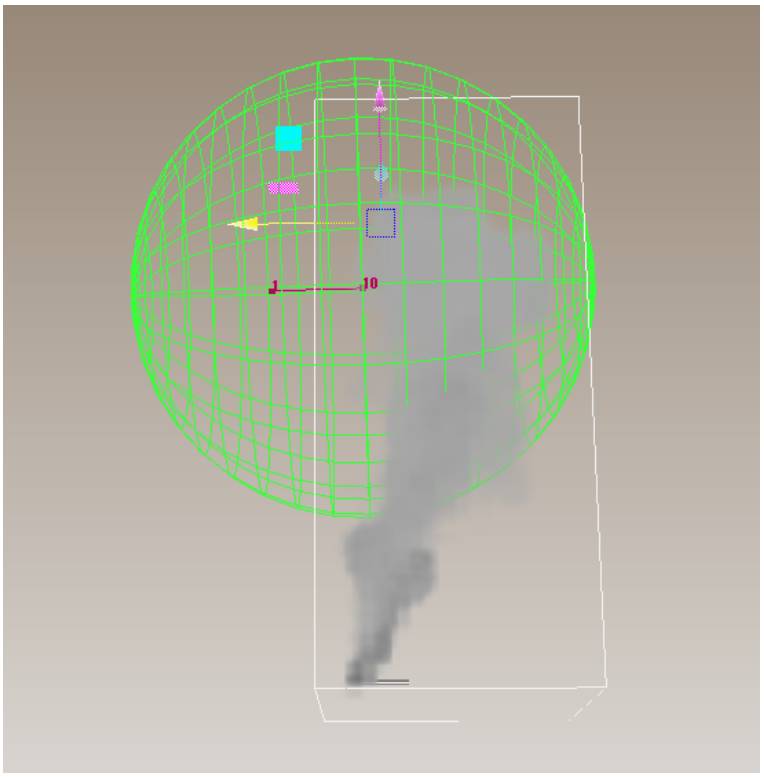


Figure 5.17: The jet particle rig in Maya, used to deform the smoke densities in the simulation. The jet particle is constrained to a curve which is placed and edited interactively. A deformed version of the densities is shown in the viewport.

The curve rig for this deformation, shown in Figure 5.17, requires a motion path in Maya defining the path of a jet particle. A deformation rig is created for easy interaction, in order to create a divergence-free flow to shape the smoke. The curve rig allows the size, particle path and momenta to be tweaked to find the ideal deformation.

By using a lower resolution proxy for the density field and OpenGL shaders, the rig updates at interactive rates enabling rapid feedback. See Section A.3.3 for more details on how the deformation rig is setup in Maya.

Although this particular deformation could have been achieved with similar results using skinning applied to volumetric data, the results of the skinning do not give incompressible flow. Moreover our setup can produce complex fluid-like deformations without painting weights or requiring bone assignments.

5.7 Conclusions and Future Work

We have proposed a method for using jet particles to edit off-line fluid simulations. We have demonstrated a system for deforming the fluid, for propagating the edit into neighbouring frames using advection, and for retracting the deformation over time for a localised effect. Smaller particles can be used to approximate turbulence-like effects and larger particles are suitable for editing the bulk flow.

Further applications and research are suggested in the following sections.

5.7.1 Coupling Rigid Bodies with Fluids

It would be interesting to use 1-jet particles as a way to couple a rigid body simulation with a fluid that had no previous interaction.

Given an off-line rigid body simulation, we can convert the positions, velocity and angular velocity of each rigid body to a constrained 1-jet particle given by (q, p, μ). This will require creating a particle with a μ that corresponds to a rigid body's angular velocity.

If we evaluate the corresponding divergence-free velocity field given by these particles, we can either augment the fluid simulation as it progresses, or use it as a post-process to deform the cached fluid flow. This latter may be useful to simulate interactions with rigid bodies that were not coupled with the fluid simulation at the time.

5.7.2 Up-Resing Existing Flows

The authors Kim et al. [41] perform an up-res process on an existing flow to add missing turbulent detail. The use of divergence-free wavelet turbulence implies the velocity projection step is not required.

The key idea here is that we can augment this framework by adding jet particles to the low resolution flow during the up-res process. Again the velocity projection step is not required since the additional velocities are divergence-free.

This extends our method of augmenting an existing flow described in Section 5.4 shown in Figure 5.10. Instead of deforming an off-line simulation with a jet particle flow, we may use live jet particles for creating a higher resolution flow.

The up-res process [41] is summarised as

- Simulate a fluid on a coarse resolution domain
- Calculate the kinetic energy of the fluid and use this to compute the wavelet decomposition
- Create a uv coordinate system based on a regular grid and advect the uv coordinates with the flow
- Create band limited, divergence-free velocity noise based on uv coordinates
- Re-simulate the fluid at higher resolution by upscaling the inflow condition, velocity, and adding the divergence-free noise to the velocity field.

Our modification to this process would add velocities given by the jet particles, which would themselves be advected with the flow. We may choose to add jet particles either representing key-frame edits, or smaller jet particles providing dynamic turbulent detail.

Chapter 6

Designing Vector Fields

We have already seen in Chapter 4 how we can design flows by constraining a jet particle's motion to a curve, and in Chapter 5 how these flows can be used to edit an existing fluid simulation. In this chapter we go on to investigate how jet particles can be used to represent existing vector fields and to design new ones.

Section 6.1 describes how to solve for the momenta of jet particles given their velocity field and its gradient. The next Section 6.2 describes how to represent existing velocity fields with jet particles in a resolution independent way.

In Section 6.3 we use configurations of jet particles to design vector fields using constraints and describe how to control the gradient of the velocity field. We show how this technique can be used to design vector fields aligned with curves for fluid control.

6.1 Projection

In this section we describe how to project a velocity field onto jet particle momenta given the velocity field and its gradient at the particle positions.

We show how to recover the momenta p_α for several 0-jet particles given their velocity u_α at their positions q_α. And we implement an algorithm that solves for the 1-jet particles p_α and μ_α given the velocity u_α and its gradient du_α at q_α. Finally we show examples of representing existing divergence-free velocity fields using this technique.

We use this method in Section 6.3 to create vector fields constrained to follow

curves.

6.1.1 Motivation

Given a known velocity field, we seek to find a configuration of jet particles which generate a velocity field that matches it.

If the given velocity field is known only at discrete points then we must take these positions into account when we choose our set of approximating jet particles.

We may have an over constrained system where we have fewer jet particles than velocity samples, and solve for the jet particles that minimise the error with respect to the velocity field. Or we may have an under constrained system when there are more jet particles than constraints if there are fewer velocity samples than jet particles.

However we choose the positions of the jet particles to be the same as the points on the known velocity grid, so we can find momenta for the jet particles that reproduce the velocity field exactly.

If our known velocity field has a continuous representation then we are free to choose a layout of jet particles. This can be a regular or a sparse set of points. However since our particles only represent frequencies at a given scale, it is important to distribute the particles so they are spaced regularly at that frequency, such that their kernels do not overlap significantly.

The obvious choices for grids are either a regular grid or an evenly spaced irregular grid. An example of the latter is a random distribution of particles generated with Poisson disk sampling [96], where new particles are added randomly using dart throwing and rejected if they are too close to existing particles than a given threshold.

A clear advantage of representing a velocity field with jet particles is that the velocity field will be divergence-free at all points since we are using incompressible matrix kernels.

6.1.2 Project Velocity Field onto 0-Jet Particles

We seek to find momenta p_α such that the velocity field $u(x)$ given by Equation (3.6)

$$u^i(x) = K^{ij}(x, q_\alpha) p_{\alpha i} \qquad \text{(3.6 revisited)}$$

matches a known velocity field $\bar{u}$ at points q_α, and interpolates a divergence-free velocity field at all other points.

Since we chose K^{ij} to be a divergence-free matrix kernel, so the vector field it generates will also be divergence-free since it is a sum of divergence-free vector fields.

We can see that the right hand side of Equation (3.6) for the velocity field $u(x)$ involves a square matrix M of size $nd \times nd$

$$[u] = [M][p] \tag{6.1}$$

and that by construction, this is a block matrix of positive semi-definite symmetric $d \times d$ matrices given by each matrix kernel K^{ij}. And further, the block diagonal of this block matrix is the identity matrix since $K(x,x) = \mathbb{I}_d$, which can be verified for our kernel by setting $r = 0$ in Equation (3.1).

As long as this matrix M is invertible, we can solve for the momenta p_α of the particles given the u_α at the particle positions since $[p] = [M]^{-1}[u]$.

When σ is zero, M is exactly the identity matrix and therfore $p_\alpha = u_\alpha$. As σ increases M becomes less invertible as the condition number increases. This is due to the increasing influence of each particle's velocity field over its neighbours.

See Listing A.2 in Section A.3.4.1 for implementation details showing code that generates the matrix M.

It is interesting to note that even if the known velocities $u(q_\alpha)$ do not come from a divergence-free velocity field, the resulting vector field $u(x)$ will always be divergence-free by construction, since it is defined as a sum of divergence-free matrix kernels.

6.1.3 Project Velocity Field onto 1-Jet Particles

We are given a velocity field $\bar{u}$ that we would like to approximate using n jet particles. As mentioned above it is not necessary that $\bar{u} \in \mathfrak{X}_{div}(\mathbb{R}^d)$. We will use n jet particles at positions $q_\alpha \in \mathbb{R}^d$ with momenta $p_\alpha \in \mathbb{R}^d$ and $\mu_\alpha \in \mathfrak{sl}_d(\mathbb{R})$. The positions q_α can be arbitrary particle positions or can be a regular grid but are assumed to be unique. It is also assumed that we know the spatial derivatives of the velocity field at q_α.

The relationship between the velocity field and the particles, given the kernel $K^{\alpha\beta}$ is

$$u^i(q_\alpha) = \qquad K^{ik}\left(q_\alpha - q_\beta\right) p_{\beta k} - \partial_k K^{il}\left(q_\alpha - q_\beta\right) \mu_\beta{}^k{}_l \qquad \text{(3.14 revisited)}$$

and so the spatial derivative of the velocity is therefore

$$\partial_j u^i(q^\alpha) = \qquad \partial_j K^{ik}\left(q_\alpha - q_\beta\right) p_{\beta k} - \partial_{kj} K^{il}\left(q_\alpha - q_\beta\right) \mu_\beta{}^k{}_l \qquad (6.2)$$

where each $\mu_\alpha \in \mathfrak{sl}_d(\mathbb{R})$ is a matrix of size $d \times d$ as before.

We would like to invert this relationship. So given velocities $\bar{u}(q_\alpha)$ and their spacial derivatives $\partial\bar{u}(q_\alpha)$ given at particle positions q_α, we will calculate the momentum p_α and reduced shape momentum μ_β that induce the same velocity and velocity gradient at q_i.

We can write Equations (3.14) and (6.2) in block matrix form as

$$\begin{bmatrix} u^i(q_\alpha) \\ \partial_j u^i(q^\alpha) \end{bmatrix} = \begin{bmatrix} K^{ik}\left(q_\alpha - q_\beta\right) & -\partial_k K^{il}\left(q_\alpha - q_\beta\right) \\ \partial_j K^{ik}\left(q_\alpha - q_\beta\right) & -\partial_{kj} K^{il}\left(q_\alpha - q_\beta\right) \end{bmatrix} \begin{bmatrix} p_{\beta k} \\ \mu_\beta{}^k{}_l \end{bmatrix}$$

or

$$\begin{bmatrix} u^i(q_\alpha) \\ \partial_j u^i(q^\alpha) \end{bmatrix} = [M] \begin{bmatrix} p_{\beta k} \\ \mu_\beta{}^k{}_l \end{bmatrix}$$

In two dimensions the right hand side contains a $6n$ square matrix M and a $6n$ column matrix contianing the unknowns p_β and μ_β. In the general case this gives a

linear system of equations with a square matrix M with $nd+nd^2$ unknown variables, where d is the dimension and n the number of particles.

When we first calculated this matrix M for a set of particles q_α, we found that the matrix had a very high condition number, even when the particles were well placed with respect to the kernel size σ. Examining the matrix in the two dimensional case, we found that the 1st and 4th row of the equations for each μ_β were linearly dependent. It became clear that the reason for this is that the matrix kernel defines a divergence-free velocity field, and hence there is a redundancy in the d^2 unknown variables μ_β. We need to include this information by modifying the system of equations, and removing the redundancy to solve for p and μ.

We remark that since each matrix μ_β is in $\mathfrak{sl}_d(\mathbb{R})$, we know that it is trace free. This implies that in two dimensions we can replace every 4th row (row $nd+4\alpha$) of the μ part of the matrix with the equation $\mu_{ii}=0$. In two dimensions this corresponds to setting $\mu^1{}_1+\mu^2{}_2=0$. This now creates a full rank matrix that can be solved for p_β and μ_β.

In three dimensions, we replace the last row for each μ_α with the equation $\mu^1{}_1+\mu^2{}_2+\mu^3{}_3=0$. See Listing A.3 in Section A.3.4.2 for an algorithm that will generate the matrix M in the general case of d dimensions.

We are not aware that any other author has solved for the momenta of 1-jet particles given a velocity field and its gradient at the time of writing.

For a given sigma, we can see from Equation (3.1) that this kernel falls off exponentially as e^{-r^2/σ^2}. So this block matrix will be sparse to the extent that the kernel K dissipates. Block cells that correspond to interactions between more distant particles will be much smaller, with the exact relationship depending on the value of σ.

6.2 Vector Field Interpolation

We have seen how jet particles can be used to construct divergence-free vector fields by summation of their matrix-valued kernels. In this section we investigate how we can approximate existing vector fields with configurations of jet particles.

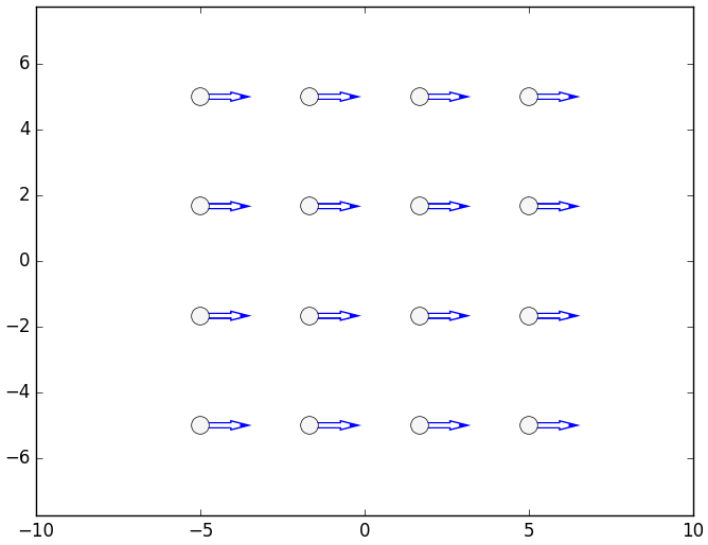
6.2.1 Examples

In order to understand how jet particles can interpolate vector fields, we choose two known vector fields and reconstruct them with both 0-jet and 1-jet particles placed on a regular grid.

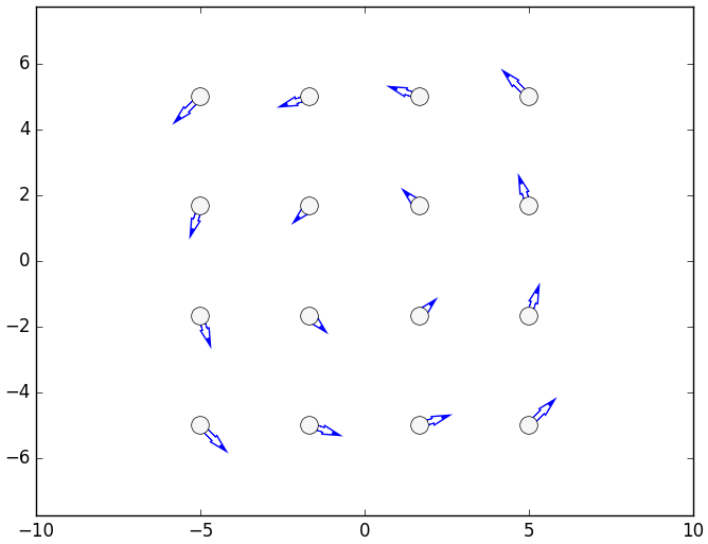
We choose a constant vector field in the x-direction shown in Figure 6.1a and a rotational divergence-free vector field, shown in Figure 6.1b.

We project these vector fields on to a regular grid of sixteen jet particles using the methods from Section 6.1 and we use the implementation from Section A.3.4 to find the particle momenta.

To illustrate the importance of the kernel size in the reconstruction, we vary the particle size σ used to evaluate the matrix kernel in the range $[0.25, 5]$.



(a) Constant vector field



(b) Rotational velocity field

Figure 6.1: The known test velocity fields

6.2.2 Results

Here we show the results of projecting the known vector fields.

Figures 6.2 and 6.3 show the 0-jet projections of the constant vector field (Figure 6.1a) and rotational divergence-free vector field (Figure 6.1b) respectively.

Figures 6.4 and 6.5 show the 1-jet projections of the constant vector field (Figure 6.1a) and rotational divergence-free vector field (Figure 6.1b) respectively. Each of these images show the reconstruction of the interpolated vector fields using only the jet particles' velocity fields.

We choose the range of sigma for each figure to include the point at which the reconstructed momenta become chaotic.

The particles are coloured so that red and blue indicate positive and negative vorticity, found by taking the anti-symmetric part of the reduced momentum μ.

6.2.3 Conclusions

Note from the figures the relationship between the kernel size σ and the spacing between the particles. It is clearly important to choose the size of σ carefully in order to reproduce the original vector field accurately.

When σ tends to zero, the particle momenta are equal to the velocities as there is no interaction between particles. If σ is too small then the vector field is only represented sparsely and cannot interpolate the target velocities in between the particle positions. As σ increases the vector field becomes resolved well in between the particles. But as σ becomes large, the condition number for the matrix to be inverted becomes higher, and numerical solutions to the momenta become less accurate as more particles begin to affect each other. This yields solutions where the momenta do not correspond to the underlying velocity field well.

So when σ is too large compared to the the particle separation, all particles interact significantly and the momenta start look random and disconnected from the underlying velocity field.

Note that by taking a divergence-free vector field and representing it with jet particles, we are able to edit the vector field by using the jet particles as handles. We create a new edited version by moving the jet particles or changing their momenta. The velocity fields produced will always be divergence-free since we use Equations (3.14) to evaluate them. Hence this represents a system for editing vector fields.

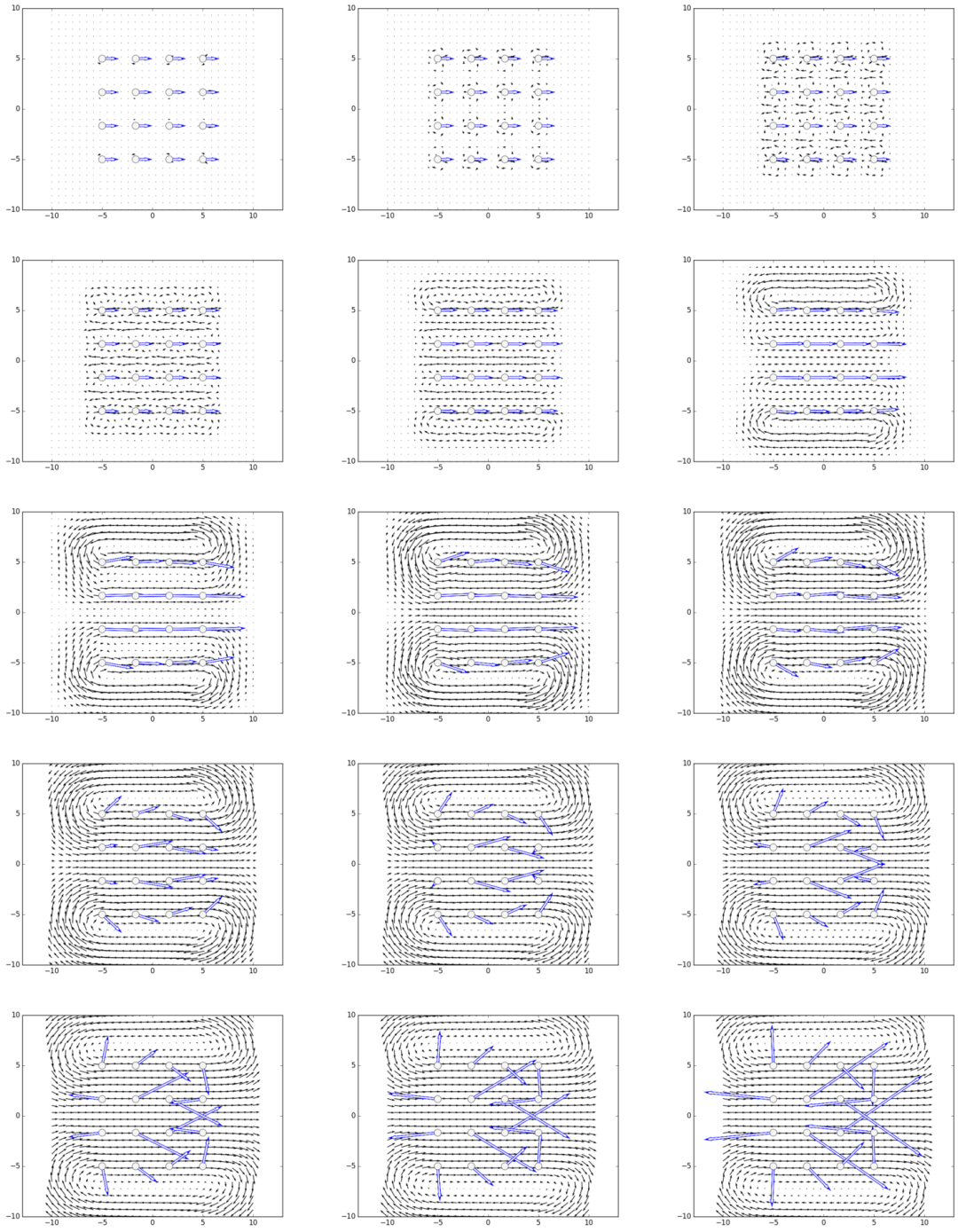


Figure 6.2: 0-Jet particles representing a constant field with $\sigma \in [0.25, 5]$

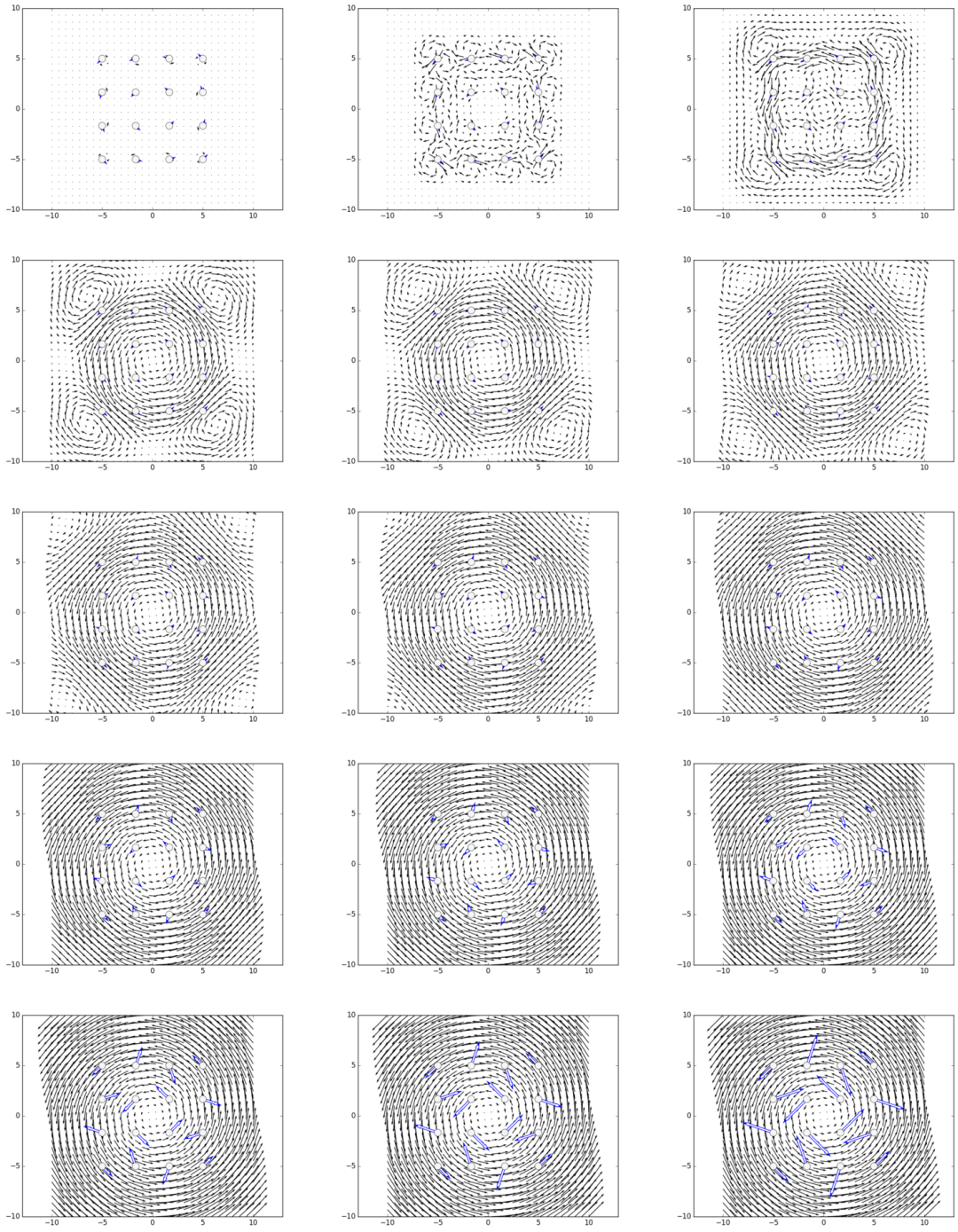


Figure 6.3: 0-Jet particles representing a divergence-free rotational field with $\sigma \in [0.25, 10]$

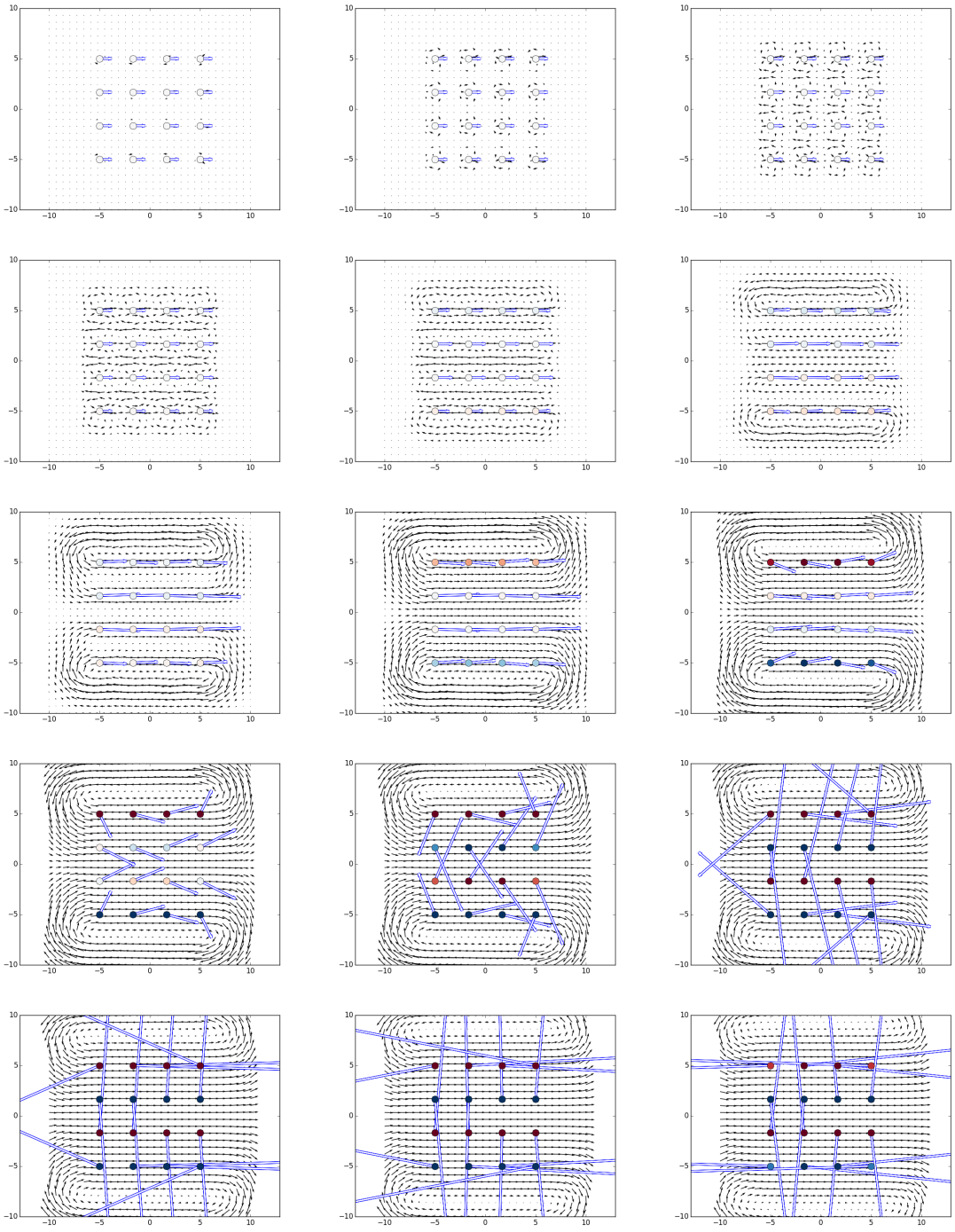


Figure 6.4: 1-Jet particles representing a constant field with $\sigma \in [0.25, 4]$

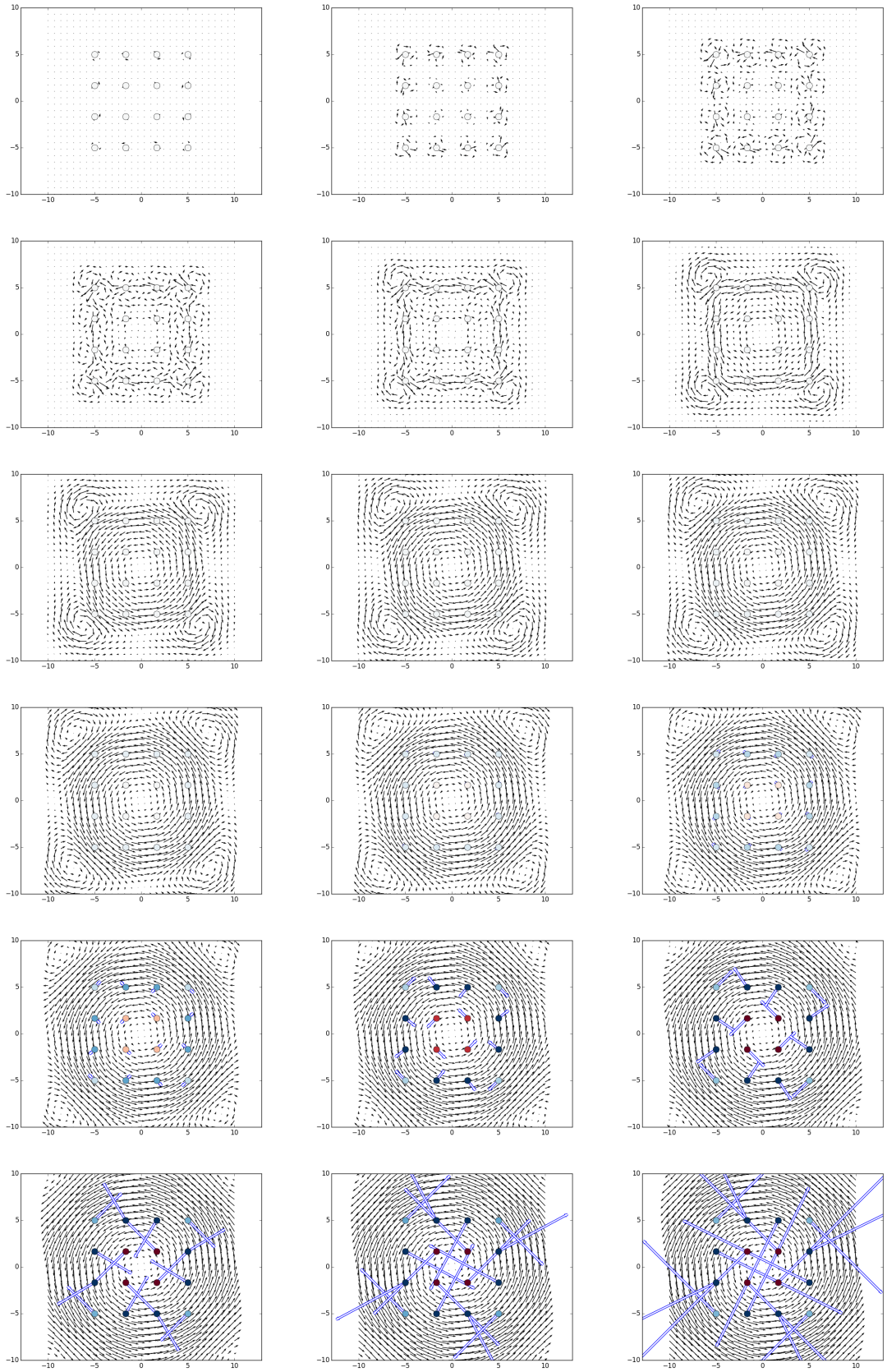


Figure 6.5: 1-Jet particles representing a divergence-free rotational field with $\sigma \in [0.25, 5]$

6.2.4 Future Work

It would be interesting to compare the interpolated velocity fields of the 0-jet and the 1-jet particles solved from the same underlying velocity fields with the same σ sized particles. For example we could generate a random divergence-free vector fields with curl noise in certain frequencies and then compare the interpolation schemes. It seems intuitive that the 1-jet velocity field interpolation will have a higher order accuracy since it is using a higher order approximation scheme.

A comparison with using scalar kernels would also be interesting, although the interpolated vector field will not be divergence-free without the use of the matrix-valued kernels, and both 0-jet and 1-jet particles should perform better.

This technique also has potential to offer compression of divergence-free velocity fields. Given an existing divergence-free velocity field stored on a grid as velocity vectors, it may be possible to represent the same velocity field to a given error tolerance with less memory using jet particles.

However for the purposes of this chapter it is sufficient to have generated velocity fields that are divergence-free, match a given velocity field at a set of points when using 0-jet particles, and additionally match the velocity gradient when using 1-jet particles. Using this machinery we can proceed to Section 6.3 in order to design useful velocity fields for use in fluid control and animation.

6.3 Vector Field Design

In computer graphics it is often useful to create vector fields for use in fluid control and animation. We will show how to design vector fields that align with curves in three dimensions using jet particles.

However, the projection method of Section 6.1 is agnostic to the positions of the jet particles, and although we demonstrate this technique on points distributed on a curve, it will work for points distributed on a regular grid in three dimensions. Hence this technique may be suitable for controlling vector fields in volumes, in a 3D analogue of the examples in Section 6.2, but we do not demonstrate this here.

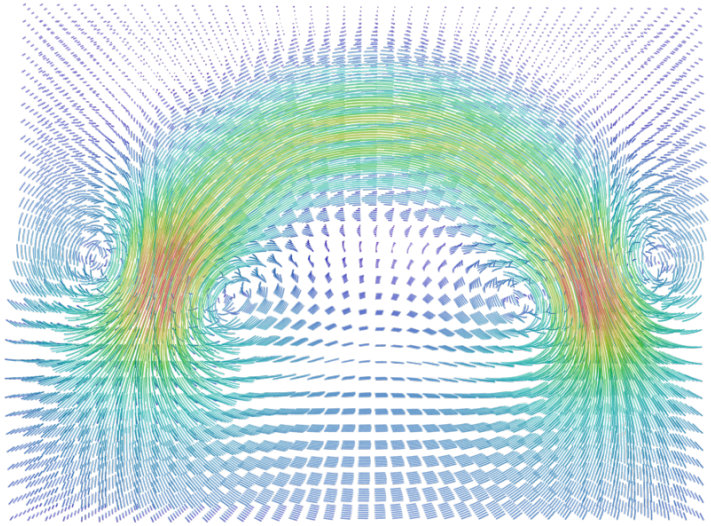


Figure 6.6: Vector field matching curve tangents, $\sigma = 0.8$

6.3.1 Tangent Constraints

If we define a space curve with a function $\gamma(s) : I \to \mathbb{R}^3$ where $I \subset \mathbb{R}$, then its velocity is given by $\gamma'(s) = \frac{d\gamma(s)}{dt}$. See Carmo [97] for an introduction to space curves in differential geometry. We would like to define a smooth divergence-free velocity field $u(x) : \mathbb{R}^3 \to \mathbb{R}^3$ such that its restriction to the image of the curve $u|_{\gamma(I)}$ is $\gamma'(s)$, so that it matches the velocity of the curve $\gamma'(s_i)$ at q_i.

We can now solve this problem using the technique from Section 6.1.2 by discretising the curve $\gamma(I)$ into n particles at positions q_i which occur at parameter values s_i, so that $\gamma(s_i) = q_i$ with $i \in \{1, \ldots, n\}$.

Figure 6.6 shows an example where a curve is discretised into seven points, and a divergence-free vector field is generated which aligns with the tangent vectors of the curve. This is achieved by solving for the momenta p_i which match the velocities $\gamma'(s_i)$ at q_i using a σ of size 0.8.

Figure 6.7 shows another example where σ is 0.2, too small to generate a visually smooth vector field away from the curve. Having the point separation scale with the kernel size σ keeps the velocity field well resolved away from the points.

We show another example with a closed curve in Figure 6.8. Here the points

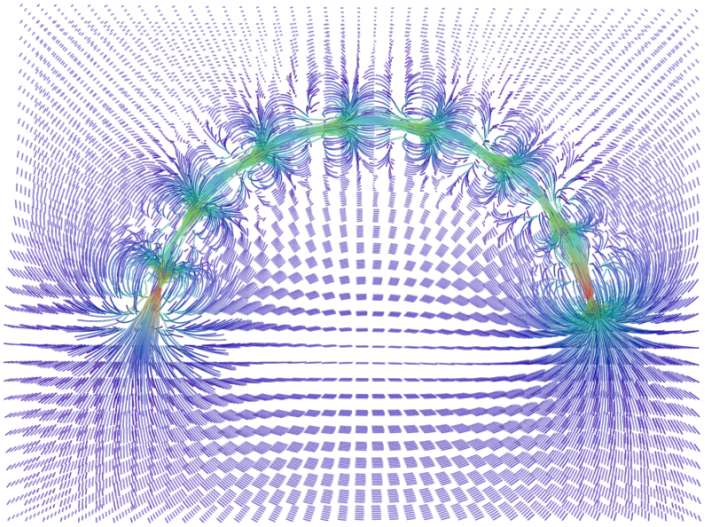


Figure 6.7: Vector field matching curve tangents, $\sigma = 0.2$

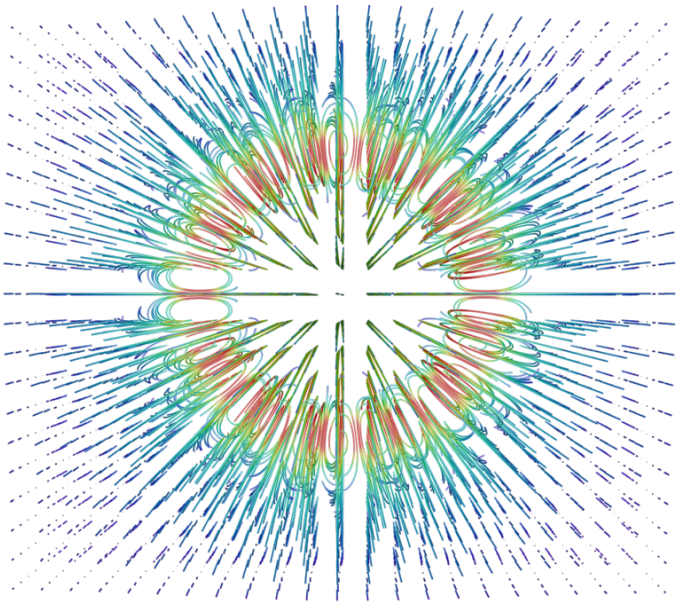


Figure 6.8: Vector field matching velocities along a closed curve

are chosen to lie on a circle, with velocities along the tangent direction.

We summarise the method for matching the velocity field to the curve tangents in Algorithm 3. Here the function TANGENTS evaluates the velocity field of the curve γ at the point q_α, while SOLVEMOMENTA solves Equation (6.1) for p_α.

Algorithm 3 Constraining velocity field

```
1: procedure CONSTRAINED VELOCITY FIELD
2:     u_α ← TANGENTS(γ, q_α)
3:     p_α ← SOLVEMOMENTA(u_α)
```

6.3.2 Deformation Gradient Constraints

If we use 1-jet particles to generate the vector field then we are free to specify an extra parameter μ which gives us control over the deformation gradient. The velocity and its gradient are given by

$$u^i(q_\alpha) = K^{ik}(q_\alpha - q_\beta)\, p_{\beta k} - \partial_k K^{il}(q_\alpha - q_\beta)\, \mu_\beta{}^k{}_l \qquad \text{(3.14 revisited)}$$

$$\partial_j u^i(q^\alpha) = \partial_j K^{ik}(q_\alpha - q_\beta)\, p_{\beta k} - \partial_{kj} K^{il}(q_\alpha - q_\beta)\, \mu_\beta{}^k{}_l \qquad \text{(6.2 revisited)}$$

as functions of p_α and μ_α.

Note that if $p_{\alpha i} = 0$ and the kernel size σ is small enough such that each particle does not influence its neighbour, then we can control each $\partial_j u^i(q^\alpha)$ directly with the corresponding $\mu_{\alpha i}{}^j$ since $\partial_{kj} K^{il}(0,0) = \delta^{il}{}_{kj}$. Note that this does not require a matrix solve since in this case $\partial_j u^i(q^\alpha) = -\mu_\alpha{}^i{}_j$ for each particle label α.

Just as in Section 4.5 where we align a matrix kernel with the brush stroke input from an artist in two dimensions, we can now choose a matrix μ in $\mathfrak{sl}_3(\mathbb{R})$. We may choose from the list of matrices in Table 4.1 representing rotation, scaling or shear or take some linear combination thereof. The choice will be motivated by the desire to rotate, scale or shear the flow along a particular axis.

Once we have chosen our μ value, we would like to align it with the curve. To guide the alignment we require an orthonormal frame. Here we can choose the Frenet frame consisting of the unit tangent, normal and binormal vectors $\{t(s), n(s), b(s)\}$ (see [97]). Alternatively we may use another frame defined along the curve, for example the Bishop's frame [98] or a user defined frame.

Since μ transforms like

$$\tilde{\mu}_i{}^j = \mu_k{}^\ell \partial_i \phi^k(q) \partial_\ell \phi^j(q)$$

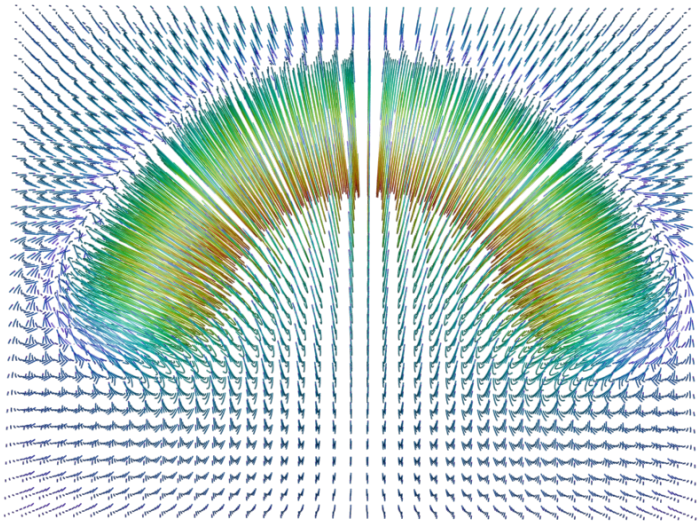


Figure 6.9: Vector field given by aligning $\mu = R_y$ with the Frenet frame

where ϕ is the change in coordinates, then if we choose a frame R for alignment then the matrix

$$\bar{\mu} = R\mu R^T$$

will align the deformation gradient given by μ such that it sits with the frame R and is tangent to the curve. We use this approach in Section 4.5 to align μ with the brush stroke.

Figure 6.9 shows the vector field resulting from aligning a μ given by R_y with the Frenet frame of the curve, in order that the vector field generates a rotation around the axis of the curve. This is in contrast to Figure 6.6 where the velocity field follows the curve, since there we are using 0-jet particles and so not able to control the deformation gradient directly.

6.3.3 Controlling the Velocity and Gradient

It is clearly useful to control the velocity as we have done in Section 6.3.1 and the deformation gradient as seen in Section 6.3.2. However if we need to do both of these at the same time then we can proceed as follows.

We calculate the momenta of 0-jet particles p_α such that the velocity u matches the space curve's derivative as before. Now we calculate the deformation gradient $\partial_j u^i$ of the velocity field of the particles with momenta p_α. Finally we add the target deformation gradient given by our control parameters μ_α to give the combined deformation gradient

$$\partial_j u^i - \bar{\mu}_\alpha{}^i{}_j$$

Now we want to solve for new p_α and μ_α such that

$$u^i(q_\alpha) = \quad K^{ik}\left(q_\alpha - q_\beta\right) p_{\beta k} - \partial_k K^{il}\left(q_\alpha - q_\beta\right) \mu_\beta{}^k{}_l \qquad \text{(3.14 revisited)}$$

$$\partial_j u^i - \bar{\mu}_\alpha{}^i{}_j = \quad \partial_j K^{ik}\left(q_\alpha - q_\beta\right) p_{\beta k} - \partial_{kj} K^{il}\left(q_\alpha - q_\beta\right) \mu_\beta{}^k{}_l \qquad (6.3)$$

As described in Section 6.1.3 we can view this equation as a square matrix of size $(nd + nd^2) \times (nd + nd^2)$ as a block matrix of matrix kernels, and we now solve for p_α and μ_α given the left hand side.

As discussed previously we have to modify the system to make sure it is invertible. Since we are in three dimensions and $\mu \in \mathfrak{sl}_3(\mathbb{R})$ it will satisfy the trace free condition $\mu_{ii} = 0$

$$\mu_{11} + \mu_{22} + \mu_{33} = 0$$

so we replace every $(3n+9\alpha+8)$th row (the row for $\mu_{\alpha 33}$) with the above condition for particle label α.

Now we can solve for a new set of jet particles that have the desired velocity and user prescribed deformation gradient at the discretised points.

Figure 6.10 shows the vector field resulting from aligning $\mu = R_y$ with the Frenet frame of the curve so that the vector field generates a rotation that twists around the axis of the curve *and* its velocity vector matches the space curve $\gamma(s)$.

It is also be natural to desire to control the local scaling perpendicular to the

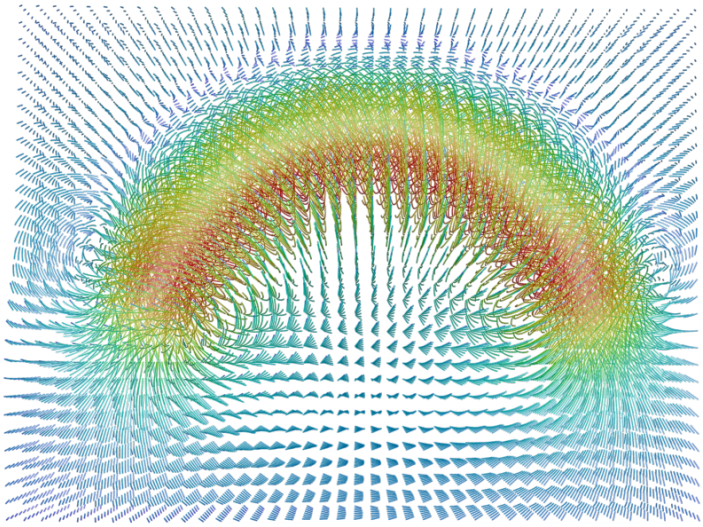


Figure 6.10: Vector field given by aligning μ with the Frenet frame and matching u on γ

tangent of the curve in order to flatten or expand the vector field along the curve.

The following Algorithm 4 summarises the steps required to create the constrained velocity field. The function DEFORMATIONGRADIENT evaluates the deformation gradient given by 0-jet particles with momenta p_α at positions q_α. The frame along the curve is provided by the function FRAME, and the 1-jet momenta are solved with the two parameter version of the SOLVEMOMENTA function that takes the additional argument of the velocity gradient.

Algorithm 4 Constraining deformation gradient and velocity field

1: **procedure** CONSTRAINED VELOCITY FIELD
2: $u_\alpha \leftarrow \text{TANGENTS}(\gamma, q_\alpha)$
3: $p_\alpha \leftarrow \text{SOLVEMOMENTA}(u_\alpha)$
4: $\partial u_\alpha \leftarrow \text{DEFORMATIONGRADIENT}(p_\alpha, q_\alpha)$
5: $R_\alpha \leftarrow \text{FRAME}(\gamma, q_\alpha)$
6: $\bar{\mu}_\alpha \leftarrow R_\alpha \mu_\alpha R_\alpha^T$
7: $p_\alpha, \mu_\alpha \leftarrow \text{SOLVEMOMENTA}(u_\alpha, \partial u_\alpha - \bar{\mu}_\alpha)$

6.3.4 Varying the Kernel Size

We may want to vary the kernel size of each particle in order to generate more interesting and varied velocity fields, similar to the vector fields used to create multi-

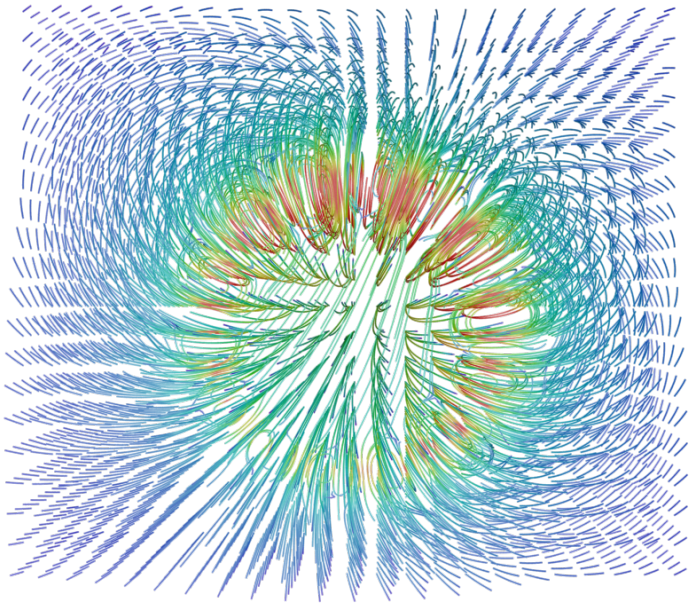


Figure 6.11: Vector field generated by 0-jet particles with varying σ_β

scale deformations in the LDDKBM framework [59]. Hence we may modify the kernels to evaluate the velocity and its gradient with the matrix kernel K_{σ_β} which now have size σ_β which vary per particle label β.

$$u^i(q_\alpha) = K^{ik}_{\sigma_\beta}(q_\alpha - q_\beta)\, p_{\beta k} - \partial_k K^{il}_{\sigma_\beta}(q_\alpha - q_\beta)\, \mu_\beta{}^k{}_l \tag{6.4}$$

$$\partial_j u^i - \bar{\mu}_\alpha{}^i{}_j = \partial_j K^{ik}_{\sigma_\beta}(q_\alpha - q_\beta)\, p_{\beta k} - \partial_{kj} K^{il}_{\sigma_\beta}(q_\alpha - q_\beta)\, \mu_\beta{}^k{}_l \tag{6.5}$$

Figure 6.11 shows a closed curve where 1-jet particles have been placed around a circle to create a velocity field that rotates around its circumference and the particle size σ has doubled on the opposite side of the circle.

Finally we show an image in Figure 6.12 of a vector field that has been created to approximate the effect of a tornado. Using only eight particles with varying kernel sizes, it has a core velocity field rising along a helix with an additional rotating component given by a 1-jet particle with a rotation only μ.

Note that since we have varied the kernel size of the particles we do not have jet particles any more, but we can still find matrix kernels that solve for the velocity and its gradient that are divergence-free.

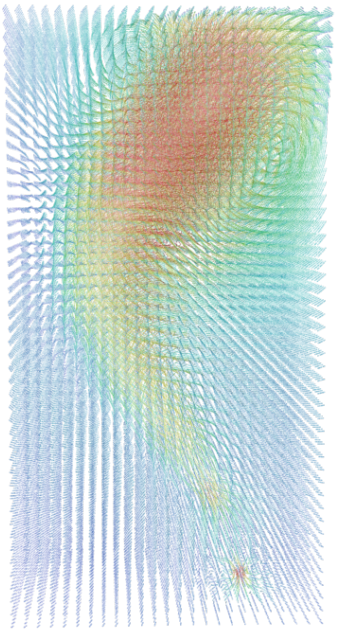


Figure 6.12: Vector field generating a tornado effect using eight 1-jet particles with varying σ_β

We believe that the tools presented in this section support our Hypothesis 3 concerning the creation and control of divergence-free vector fields.

6.4 Conclusions and Future Work

6.4.1 Representing Existing Velocity Fields with Jet Particles

One application of the techniques explored in this chapter is to compress the velocity vector fields from a cached simulation with jet particles.

The reduction of existing flows to sparse data sets such as vortex filaments [34] offers opportunities for compression and creating reduced solvers, and a hierarchical versions of this approach provides a method for editing velocity fields in a natural way [35].

We believe that representing existing time-varying velocity fields as jet particles can offer similar benefits. Given a velocity field from a incompressible fluid simulation, we can represent the velocities using jet particles by projecting the velocity field onto particles momenta. Farrell, Gillow, and Wendland [99] develop a multi-level technique for interpolating divergence-free vector fields from scattered

data points using divergence-free matrix kernels, so there is a precedence for this approach.

We would like to investigate how to compress vector fields from production fluid simulations with jet particles using the matrix kernels we have discussed [62] [36] in Chapter 3. We would seek to minimise the reconstruction error [30] of the full vector field given the jet particles at given positions.

The jet particles can be kept on a static grid which is simplest as it keeps a good sampling resolution. Alternatively the particles may move due to their equations of motion and at each time step we can project the velocity fields again onto the particles. This may provide a natural temporal basis for fluid editing by using jet particles as handles.

6.4.2 Designing Flows on Surfaces

Another interesting topic is designing flows on surfaces. We may scatter jet particles on a surface and interpolate their velocity field anywhere on the surface, providing that the points are well placed with respect to the kernel size. There is also the added advantage that the velocities can be trivially extended into the volume in a similar way that Bhattacharya, Nielsen, and Bridson [100] extrapolate flux velocities into the interior of a surface.

6.4.3 Jet Particle Flow Rigs

In order to design flows easily for computer games or animation, it may prove useful to cache an existing flow onto a curve rig or grid over a period of time. Having recorded the dynamics of the fluid onto the rig for some time interval, we are free to treat the curve as a skeletal rig and animate it in order to create a new flow. The momenta of the jet particles may be pushed forward with the animation of the curve rig, and the original simulation may be played back in a new pose.

The basic idea is summarised as:

1. Cache a fluid simulation $u(x,t)$
2. Project the simulation onto a 1-jet particle grid (p_α, μ_α)
3. Design a secondary edit flow φ e.g. with a jet particle curve rig or skeletal

animation

4. Find the push forward of the jet particle rig $(\varphi(q_\alpha), \varphi_*(p_\alpha), \varphi_*(\mu_\alpha))$
5. Calculate the new deformed velocity field

Chapter 7

Deformations with Jet Particles

In this chapter we develop efficient methods for realising the coordinate change given by the jet particle flow. Here we develop the machinery that allows us to implement the deformations we require in Chapters 4 and 5.

The first algorithm we develop in Section 7.1 allows us to deform the density and also evaluate the push-forwards of the velocity vector with the flow. This is useful for example to calculate motion blur for use during rendering. We improve on the accuracy of the Jacobian calculation in Section 7.2 by using the reverse jet particle flow on the dense target grid.

In Section 7.3 we reduce computation times significantly by using a coarse grid to accelerate the direct approach, ensuring we are able to process the large grids produced by current simulators. The accuracy of this method is improved further by using cubic bezier interpolation in Section 7.4, enabling the use of still coarser deformation grids.

Finally in order to create deformations from jet particle simulations not initialised on a regular grid, we develop a gridless deformation algorithm in Section 7.5.

7.1 Forwards Deformation

The first method developed here to deform fluid densities and velocities based on jet particle flow was implemented in two dimensions in the software *mantaflow* [91].

The fluid domain is discretised into a regular grid and its vertices are flowed

forward using the velocities induced by the jet particles. The velocities for each grid point are calculated using the function EVALUATEVELOCITY defined in Algorithm 5, by accumulating the velocities from each jet particle calculated using Equation (3.14).

Algorithm 5 Evaluate velocity

1: **function** EVALUATEVELOCITY(J, x)
Input: Jet Particles $J = \{j_1, \ldots, j_n\}$
Input: Position to evaluate $x \in \mathbb{R}^d$
Output: Velocity u
2: $u \leftarrow 0$
3: **for** $\alpha \in \{1, \ldots, n\}$ **do**:
4: $(q_\alpha, p_\alpha, \mu_\alpha) \leftarrow$ Jet Particle j_α
5: $u \leftarrow u + u(x, q_\alpha, p_\alpha, \mu_\alpha)$ Equation (3.14)

The flow of the vertices are calculated by time-stepping using forwards Euler integration. Higher order techniques can be used for more accuracy.

Algorithm 6 Advection with flow from jet particles

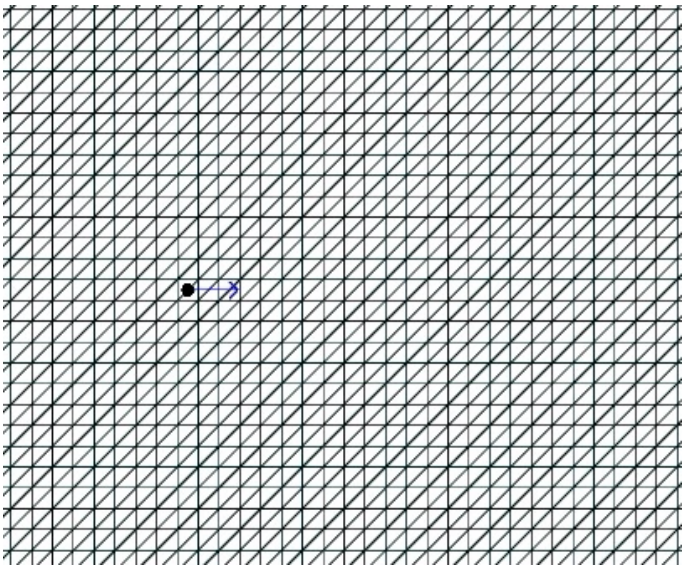
1: **function** FLOWFORWARDS
Input: Jet Particles $J(t) = \{j_1(t), \ldots, j_n(t)\}$ at times $t \in \{t_1, \ldots, t_N\}$
Input: Position to flow $x \in \mathbb{R}^d$
Output: Deformed position x
2: **for** $t \in \{t_1, \ldots, t_N\}$ **do**:
3: $u \leftarrow$ EVALUATEVELOCITY$(J(t), x)$
4: $x \leftarrow x + u\,dt$

See Figure 7.1 showing the deformation of a regular grid from its rest domain into deformed elements via the flow of a single 0-jet particle with momentum in the x direction.

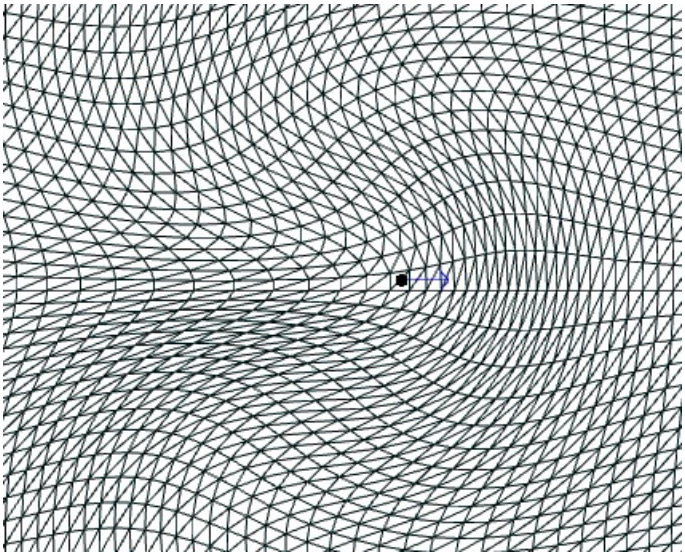
In order to use these grids to find the deformed densities and velocities from the rest grid we proceed as follows:

For each deformed position p we find its containing triangle element T and calculate the barycentric coordinates λ_i of p within T. We then find the rest position p_R for p by using the barycentric coordinates λ_i within the rest triangle element T_R.

In order to push forward the velocity vector u_R from the rest grid, we calculate the Jacobian of the deformation in the triangle. This is done by finding the unique



(a) Rest grid of trianglulated elements



(b) Grid deformed with 0-jet particle flow

Figure 7.1: Deformation of rest grid with a 0-jet particle

$d \times d$ matrix M that transforms the rest triangle T_R into T, after translating each triangle such that a corresponding reference vertex is at the origin. See Figure 7.2 showing how we approximate the push forward φ^* with M.

Then the push forward of the rest velocity is just

$$u = M \cdot u_R$$

This method is summarised in Algorithm 7 as the function PUSHFORWARD.

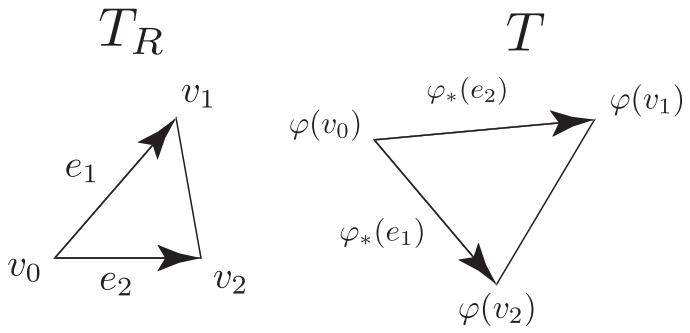


Figure 7.2: The Jacobian matrix maps the rest triangle T_R to the triangle T deformed by the flow φ

Algorithm 7 Deform Density and Velocity

function PUSHFORWARD
Input: Position $p \in \mathbb{R}^d$ in deformed state
Input: Rest grid R carrying density d_R and velocity u_R values
Input: Deformed grid G
Output: Deformed density d at p
Output: Deformed velocity vector u at p
$T \leftarrow$ Triangle containing p
$\lambda_i \leftarrow$ Barycentric coordinates of p in T
$v_i \leftarrow$ Vertices of rest triangle T_R
$p_R \leftarrow$ Rest point position $\sum \lambda_i v_i$
$d_i \leftarrow$ Density values at triangle vertices v_i
$d \leftarrow$ Density $\sum_i \lambda_i v_i$
$M \leftarrow$ Jacobian $d \times d$ matrix that maps T_R to T without translation
$u_i \leftarrow$ Rest velocity values at triangle vertices v_i
$u_R \leftarrow$ Interpolated rest velocity $\sum_i \lambda_i u_i$
$u \leftarrow$ Push forward $M \cdot u_R$

The draw back of this method is that it requires us to maintain an acceleration data structure to track which deformed triangle element T contains each point p. Moreover the interpolation is only piecewise linear for the density within each triangle, and only piecewise constant for the velocity, since the Jacobian matrix is fixed for each triangle.

7.2 Backwards Flow

The idea here is to flow each point of interest back in time in order to find the pre-image of the point under the flow, so we can look up the interpolated quantities in the source grid directly.

To this end we create a new function EVALUATEDERIVATIVES in Algorithm 8 which allows us to calculate the change in the deformation gradient Q due to the jet particles flow. Using this we create an updated function FLOWFORWARDS in Algorithm 9 which integrates position and a frame describing how the fluid is being deformed locally. We also create the corresponding function FLOWBACKWARDS in Algorithm 10 which will solve for the reverse flow.

Algorithm 8 Evaluate derivatives

1: **procedure** EVALUATEDERIVATIVES(J,x)
Input: Jet Particles $J = \{j_1,\ldots,j_n\}$
Input: Position evaluate $x \in \mathbb{R}^d$
Output: Velocity u
Output: Time derivative of deformation gradient dQ
2: $u \leftarrow 0$
3: $dQ \leftarrow 0_d$
4: **for** $\alpha \in \{1,\ldots,n\}$ **do**:
5: $(q_\alpha, p_\alpha, \mu_\alpha) \leftarrow$ Jet Particle j_α
6: $u \leftarrow u + u(x, q_\alpha, p_\alpha, \mu_\alpha)$ Equation (3.14)
7: $dQ \leftarrow dQ + \frac{\partial u^i_\alpha}{\partial x^j}$ Equation (3.9)

Note the gradient $\frac{\partial u^i_\alpha}{\partial x^j}$ is calculated using Equation (3.9).

Algorithm 9 Advection with flow from jet particles

1: **procedure** FLOWFORWARDS
Input: Jet Particles $J(t) = \{j_1(t),\ldots,j_n(t)\}$ at times $t \in \{t_1,\ldots,t_N\}$
Input: Position to flow $x \in \mathbb{R}^d$
Output: Deformed position x
Output: Deformation gradient Q
2: $Q \leftarrow \mathbb{I}_d$
3: **for** $t \in \{t_1,\ldots,t_n\}$ **do**:
4: $(u, dQ) \leftarrow$ EVALUATEDERIVATIVES$(J(t), x)$
5: $x \leftarrow x + u\,dt$
6: $Q \leftarrow Q + dQ \cdot Q\,dt$

The function FLOWBACKWARDS differs from FLOWFORWARDS in that that

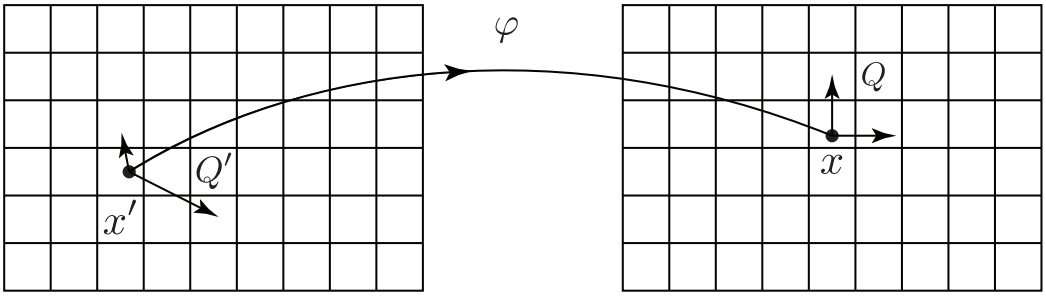


Figure 7.3: Flow by φ mapping x' to x

the Euler integration step uses the negative velocity and gradient to integrate over the time steps in reverse.

Algorithm 10 Reverse advection with flow from jet particles

1: **procedure** FLOWBACKWARDS
Input: Jet Particles $J(t) = \{j_1(t), \ldots, j_n(t)\}$ for t in $t_1, \ldots, t_N$
Input: Position to flow $x \in \mathbb{R}^d$
Output: Deformed position x
Output: Deformation gradient Q
2: $Q \leftarrow \mathbb{I}_d$
3: **for** $t \in \{t_n, \ldots, t_1\}$ **do**:
4: $(u, dQ) \leftarrow$ EVALUATEDERIVATIVES$(J(t), x)$
5: $x \leftarrow x - u\,dt$
6: $Q \leftarrow Q - dQ \cdot Q\,dt$
7: **return** x, Q

For each position in the deformed grid where we wish to calculate the deformed density and velocity vector, we initialise a unit frame and flow it backwards in time to find the corresponding rest position and frame. We then use this rest position to look up the density, and we use the deformed frame to push-forward the velocity. See Figure 7.3 for an illustration.

Note that since the jet particle system is Hamiltonian, the flow is reversible. And since we are using the incompressible matrix kernels, the matrix Q' will be in $\mathrm{SL}(d;\mathbb{R})$ so it is volume preserving.

If the identity matrix $(x, \mathbb{I}_d)$ has the pre-image (x', Q') under this flow, we can look up scalar quantities such as density by evaluating the field DENSITY(x') di-

rectly and use Q' to evaluate the deformed velocities of the original flow.

Note that the transformation law for densities requires that we scale the result using the determinant of Jacobian. We can omit this step since the Jacobian will have determinant 1 due to the incompressible nature of the flow. This will always be the case when we have a flow from particles that use the incompressible matrix kernels.

If we have a vector field that we seek to deform with our edit φ (such as the velocity field from an off-line fluid simulation), we can push it forward with the edit flow as follows. The pre-image Q of the identity frame $\mathbb{I}_d$ is just the inverse of the Jacobian of the flow φ at x'. So given $J = Q'^{-1}$ we can calculate the deformed velocity vector as $J \cdot u$. Since Q' and J are both determinant 1, the push forward of the velocity field is guaranteed to keep the deformed vector field incompressible.

The advantage of this method compared to the previous method in Section 7.1 is that the density can be looked up at the exact point it was flowed from, rather than interpolating it within a deformed element. Also the deformed velocity is now smooth.

7.3 Coarse Grid Optimisation

As an alternative to evaluating every point in the deformed space, we will instead evaluate the deformation only on a coarse grid. Then the deformed position can be calculated by taking a linear combination of the estimates from each frame.

Given that we have a deformed region $\mathscr{D}$ at time t given by some flow $\varphi(\cdot,t)$ that we want to evaluate numerically, we discretise $\mathscr{D}$ into a regular *deform grid* with points X_{ij}. We also define unit frames $\mathbb{I}_d$ at each point X_{ij}. Using the function FLOWBACKWARDS defined in Algorithm 10, we flow back the grid points X_{ij} and their frames to rest time $t = 0$.

For each point q in $\mathscr{D}$ that we would like to evaluate, we find ourselves in a cell C of the *deform grid*. At the vertices X_i of this cell C (where $i \in \{1,\ldots,2^d\}$) we also know how the unit frames are deformed, since we know their push forwards Q_i from above. Hence we can approximate the reverse flow $\varphi(\cdot,t)^{-1}$ by finding the 2^d

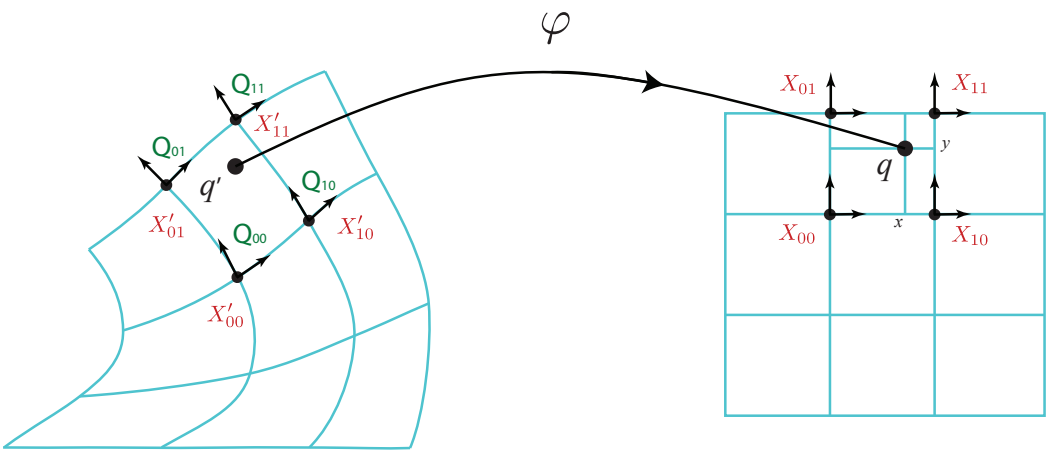


Figure 7.4: Bilinear interpolation to estimate undeformed position q' from q using frames Q_{ij} and positions X_{ij}

different approximations of the flow and blending between the results.

We calculate the blend weights using bilinear or trilinear interpolation (when d is 2 or 3 dimensions respectively) since our *deform grid* is a regular axis aligned grid.

Figure 7.4 shows how we find the bilinear coordinates in the *deform grid* to estimate the undeformed position q' using the frames Q_{ij}. The local coordinates (x,y) shown are normalised to lie in the range $[0,1]\times[0,1]$.

We transform q once for each vertex X_{ij} of the cell using the matrix Q_{ij} to get 2^d estimates q_{ij} of $\varphi(\cdot,t)^{-1}(q)$

$$q'_{ij} = X'_{ij} + Q_{ij}\cdot\left(q - X_{ij}\right) \tag{7.1}$$

We then use bilinear or trilinear interpolation of the q'_{ij} using weights based on the position of x in the cell C. So in two dimensions we approximate q with bilinear interpolation using

$$q = (1-x)(1-y)q'_{00} + x(1-y)q'_{10} + (1-x)y\,q'_{01} + xy\,q'_{11}$$

and with a similar result in three dimensions based on the interpolants (x,y,z), using trilinear interpolation.

In order to calculate the deformed velocity at q, we push forward the velocity in the source grid at each vertex with the frames Q_{ij} to evaluate the velocity on the *deform grid*. We then interpolate the deformed velocity with the same bilinear or trilinear interpolation scheme. The interpolation of the velocity may result in the deformed velocity not being divergence-free within each cell.

It is also possible to calculate an interpolated frame Q with the same interpolation method in the source grid

$$Q = (1-x)(1-y)Q'_{00} + x(1-y)Q'_{10} + (1-x)yQ'_{01} + xyQ'_{11}$$

and use this frame to push forward an interpolated velocity vector. However this interpolated matrix Q will not in general lie in $\mathrm{SL}(d;\mathbb{R})$. Hence the push forward $Q \cdot u$ may not be divergence-free in the interior of the cell.

These weights do not take any non-linearity of the deformation gradient into account and so there may be better choices. However they give reasonable results, are efficient to calculate, and are easy to implement.

This algorithm is summarised here in Algorithm 11 for the two dimensional case.

7.4 Bezier Interpolation

A higher order approximation for deformation by the coarse *deform grid* is achieved by using bezier spline interpolation. Since we know the deformation gradient at the coarse grid points, we can use this information to generate control points for cubic bezier interpolation that matches the deformation gradient.

The unit frames which have been deformed by the jet particle flow in Section 7.3 can be used to generate control points for a bezier cubic spline as in Figure 7.5. It is natural to use a *cubic* spline since we have evaluated the deformation gradient matrix at each point. This provides us with the information to create the tangents that are the both necessary and sufficient to define the control points required for

Algorithm 11 Evaluate deformed position and frame using bilinear interpolation

Input: Deform grid Φ
Input: Point position p
Output: Deformed position q
Output: Deformed frame Q

function SAMPLEDEFORMGRID(G, p)
  $c \leftarrow$ Cell containing p
  $X_{ij} \leftarrow$ Vertices of c
  $(x, y, z) \leftarrow$ Normalised coordinates of p in c
  $X'_{ij} \leftarrow$ Deformation points at X_{ij}
  $Q'_{ij} \leftarrow$ Deformation frames at X_{ij}
  $q'_{ij} \leftarrow X'_{ij} + Q'_{ij} \cdot (q - X_{ij})$
  $q \leftarrow (1-x)(1-y)q'_{00} + x(1-y)q'_{10} + (1-x)y\,q'_{01} + xy\,q'_{11}$
  $Q \leftarrow (1-x)(1-y)Q'_{00} + x(1-y)Q'_{10} + (1-x)y\,Q'_{01} + xy\,Q'_{11}$
  return q, Q

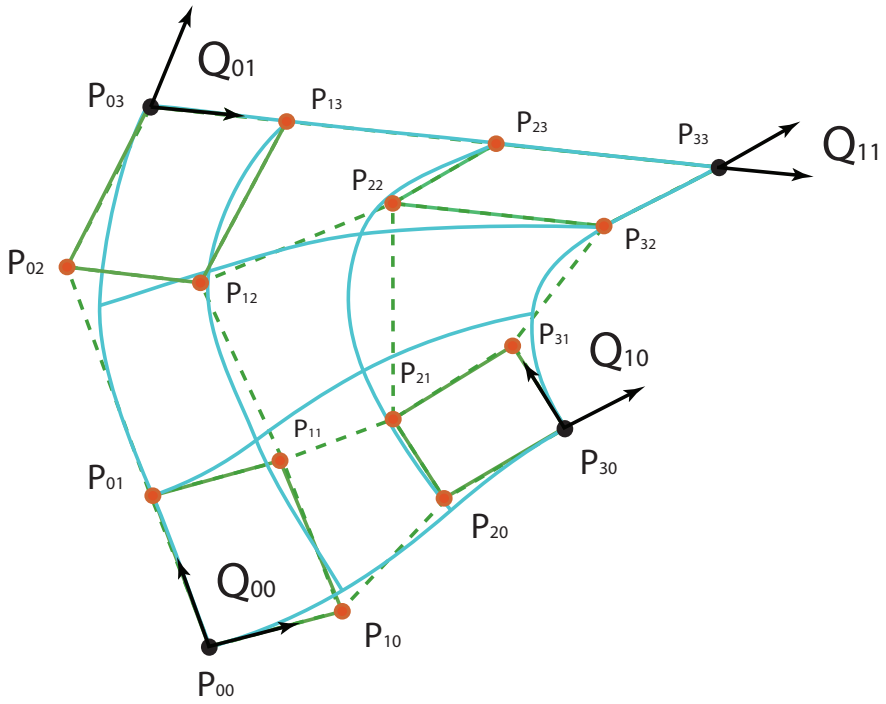


Figure 7.5: Control points for a single bezier patch in 2D given the reverse flow $\varphi(\cdot, t)^{-1}$

the Bezier spline volume.

Each cell in the coarse grid defines a cubic Bezier volume patch, and together they create a cubic Bezier volume spline for the whole deformation. The method of

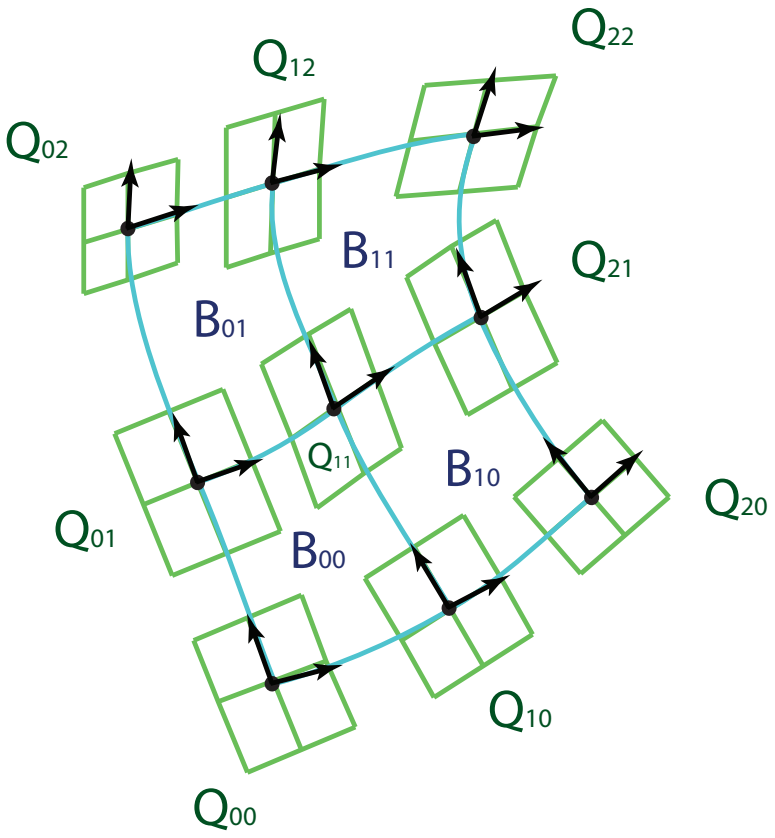


Figure 7.6: B-Spline surface consisting of four surface patches B_{ij} constructed with frames Q_{ij}

construction guarantees that neighbouring patches have matching tangents, and the B-Splines are guaranteed to have C^2 continuity even where they join neighbouring patches [101].

Figure 7.6 shows the a 2D cubic Bezier spline constructed from frames Q_{ij} calculated on a *deform grid*. We define the corner vertices of the Bezier patch $p_{00}, p_{30}, p_{03}, p_{33}$ to be the positions $X'_{00}, X'_{10}, X'_{01}, X'_{11}$ defined by the reverse flow of the cell C respectively with $\varphi(\cdot,t)^{-1}$ as before.

Now we define the 16 vertices as follows (here we just define the lower two

rows of the control points as the other rows are similar).

$$p_{00} = X'_{00} \tag{7.2}$$

$$p_{10} = X'_{00} + \frac{1}{3} Q_{00} e_x \tag{7.3}$$

$$p_{20} = X'_{10} - \frac{1}{3} Q_{10} e_x \tag{7.4}$$

$$p_{01} = X'_{00} + \frac{1}{3} Q_{00} e_y \tag{7.5}$$

$$p_{11} = X'_{00} + \frac{1}{3} Q_{00} e_x + \frac{1}{3} Q_{00} e_y \tag{7.6}$$

$$p_{21} = X'_{10} - \frac{1}{3} Q_{10} e_x + \frac{1}{3} Q_{00} e_y \tag{7.7}$$

$$p_{31} = X'_{10} + \frac{1}{3} Q_{00} e_y \tag{7.8}$$

where e_x, e_y are unit vectors in the x and y directions. These define 4 parallelograms based on each frame Q_{ij}.

A similar calculation holds to compute the 64 control points p_{ijk} required for the Bezier volume patch in three dimensions for example

$$p_{000} = X'_{00} \tag{7.9}$$

$$p_{100} = X'_{00} + \frac{1}{3} Q_{000}\, e_x \tag{7.10}$$

$$p_{110} = X'_{00} + \frac{1}{3} Q_{000}\, e_x + \frac{1}{3} Q_{000}\, e_y \tag{7.11}$$

$$p_{111} = X'_{00} + \frac{1}{3} Q_{000}\, e_x + \frac{1}{3} Q_{000}\, e_y + \frac{1}{3} Q_{000}\, e_z \tag{7.12}$$

and so on, where we are using the unit vectors e_x, e_y, e_z to calculate parallelepipeds based on each frame Q_{ijk}.

The values of $1/3$ are chosen since in the case where the corner points X'_{ij} lie on a unit square and the frames Q_{ij} are identity matrices, then it can be shown easily that the spline function corresponds to the identity map.

A point p in the Bezier surface patch can be evaluated with

$$p = \sum_{i=0}^{3} \sum_{j=0}^{3} p_{ij} B_{3,i}(x) B_{3,i}(y)$$

and the corresponding volume patch is evaluated with

$$p = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} p_{ijk} B_{3,i}(x) B_{3,j}(y) B_{3,k}(z)$$

Here the $B_{n,i}$ are the Bernstein basis polynomial functions [102] are defined as

$$B_{n,i}(x) = \binom{n}{i} x^i (1-x)^{n-i} \qquad\qquad i = 0, \ldots, n$$

and for cubic splines we have

$$\begin{aligned} B_{3,0}(x) &= (1-x)^3 \\ B_{3,1}(x) &= 3x(1-x)^2 \\ B_{3,2}(x) &= 3x^2(1-x) \\ B_{3,3}(x) &= x^3 \end{aligned}$$

Finally we can calculate an interpolated frame $Q(x,y,z)$ in the volume using the derivatives

$$\begin{aligned} \frac{\partial p}{\partial x} &= \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} p_{ijk} B'_{3,i}(x) B_{3,j}(y) B_{3,k}(z) \\ \frac{\partial p}{\partial y} &= \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} p_{ijk} B_{3,i}(x) B'_{3,j}(y) B_{3,k}(z) \\ \frac{\partial p}{\partial z} &= \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} p_{ijk} B_{3,i}(x) B_{3,j}(y) B'_{3,k}(z) \end{aligned}$$

where $B'_{3,k}(x)$ is the derivative of $B_{3,k}(x)$, to calculate the deformation gradient

$$Q = \frac{\partial p_i}{\partial x_j}$$

See more discussion about calculating derivatives of bezier surfaces in Spitzmüller [103].

Note that by evaluating these partial derivatives at $(0,0,0)$ we have

$$\frac{\partial p}{\partial x}(0,0,0) = 3(p_{100} - p_{000})$$
$$\frac{\partial p}{\partial y}(0,0,0) = 3(p_{010} - p_{000})$$
$$\frac{\partial p}{\partial z}(0,0,0) = 3(p_{001} - p_{000})$$

When we substitute for the control points with our definitions from Equation (7.9) and (7.10) and so on, we have that

$$\frac{\partial p}{\partial x}(0,0,0) = Q_{000}\, e_x$$
$$\frac{\partial p}{\partial y}(0,0,0) = Q_{000}\, e_y$$
$$\frac{\partial p}{\partial z}(0,0,0) = Q_{000}\, e_z$$

and so we find that

$$Q(0,0,0) = Q_{000}$$

So we recover the original frame as required at $(0,0,0)$ and the same will hold at the other corners, and so $Q(x,y,z)$ will interpolate the frame consistently within the volume patch.

This matrix Q may not lie in $\mathrm{SL}(d;\mathbb{R})$ in general, but we can use it to push forward velocity vectors into the deformed volume from the rest grid.

This algorithm is summarised here in Algorithm 11 for the two dimensional case.

Figure 7.7 shows control points for a bezier volume spline in 3D, constructed from a *deform grid* based on the deformation by a 1-jet particle.

7.5 Point Cloud Deformation

In Section 7.3 it is a pre-requisite that the deform grid is a regular grid. However this method can be easily modified to work for a point cloud with no grid structure. This

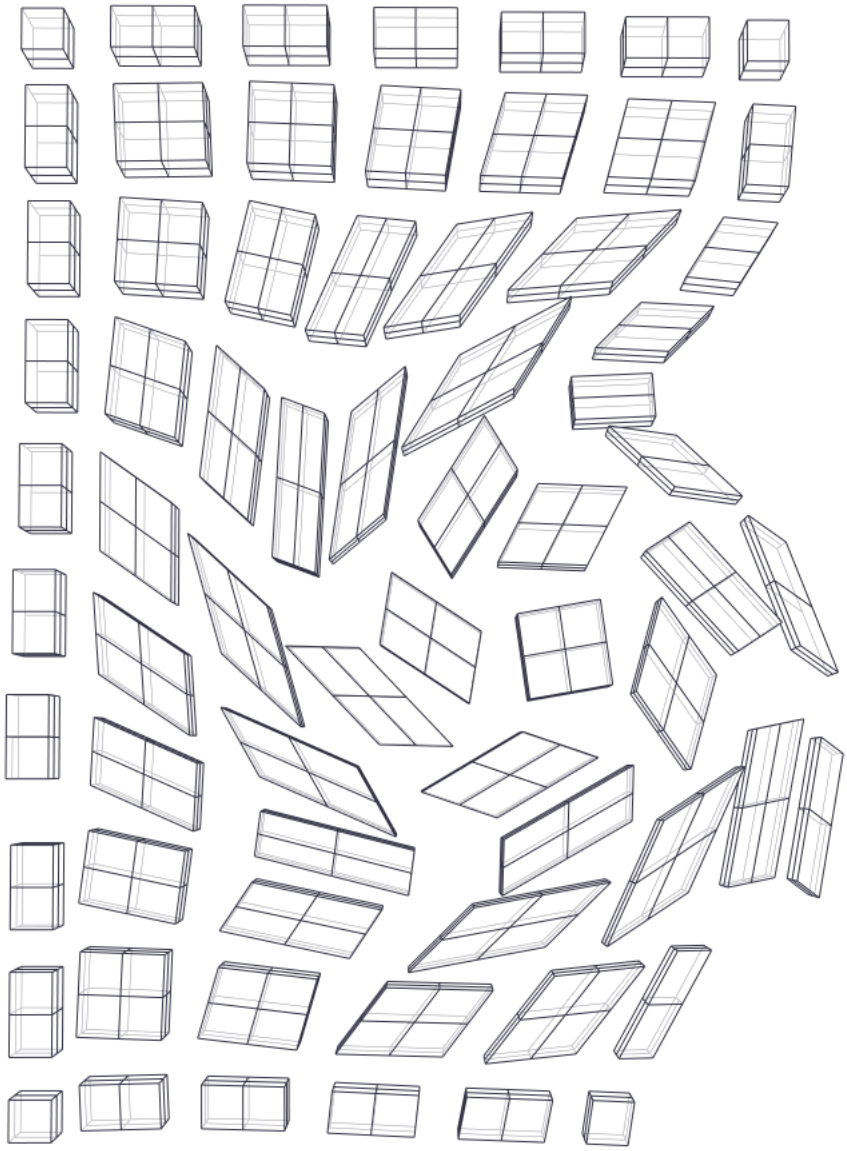


Figure 7.7: Control points for a slice of a 3D bezier spline

Algorithm 12 SampleDeformGridBezier

Input: Deform grid Φ
Input: Point position p
Output: Deformed delta dp
Output: Deformed frame Q

function SAMPLEDEFORMGRID(G, p)
  $c \leftarrow$ Cell containing p
  $X_{ij} \leftarrow$ Vertices of c
  $(x, y, z) \leftarrow$ Normalised coordinates of p in c
  $X'_{ij} \leftarrow$ Deformation points at X_{ij}
  $Q'_{ij} \leftarrow$ Deformation frames at X_{ij}
  $p_{ij} \leftarrow$ Control points using Equations (7.2)
  $q_i \leftarrow \sum_{i=0}^{3} \sum_{j=0}^{3} p_{ij} B_{3,i}(x) B_{3,i}(y)$
  $Q \leftarrow \frac{\partial q_i}{\partial x_j}$
  return q, Q

is useful to calculate deformations between frames other than the initial deformation frame, using the same jet particle flow data.

With this method, a jet particle flow is calculated from a regular grid and integrated with forwards flow as with the coarse grid, but now each deformed frame need not evaluate the reverse flow for each frame.

To calculate the deformation of a point with respect to the flow of an arbitrary point cloud, we proceed as follows. Consider a set of *deform particles* $\alpha_i := (X_i, Q_i) \in \mathbb{R}^d \times \mathrm{SL}(d;\mathbb{R})$ initialised on a regular grid with an identity frame and advected with some jet particle flow J_i over time steps $t_a, \ldots, t_b$. The result is a set of moving particles X_i that carry a deforming frame Q_i over time.

We would like to apply the deformation implied by these particles over a subset of the time steps $t_1, \ldots, t_n$ (where $t_a < t_1 < t_n < t_b$), but without running a new jet particle simulation from a regular grid.

To proceed, for each point p to deform we find the set of K closest deform particles and attach p to each deforming frame to estimate its new position. Finally as in Section 7.3, we take a weighted linear combination of each estimate, using blend weights based on a one over distance squared fall off. The process is described in Algorithm 13 and illustrated in Figure 7.8.

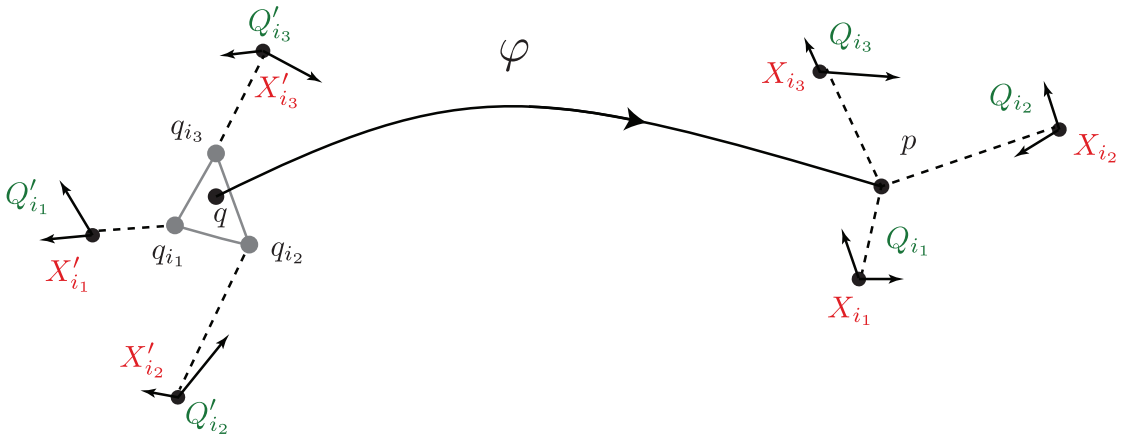


Figure 7.8: Deforming a point p with a flow of a point cloud given by closest *deform particles* $\alpha_i := (X_i, Q_i)$

The estimate for the deformed position relative to each particle α_i is

$$q_i = Q'_i Q_i^{-1}(p - X_i) + X'_i$$

which is modified from Equation (7.1) to include the inverse of the initial matrix Q_i, not the identity matrix.

In a similar way that the skinning algorithm uses bones and blend weights (see Section 5.1.1) to approximate a deformation, we use our frames Q_i as bones and calculate suitable weights. The difference is that our bones will not be orthogonal matrices since they have been deformed with the flow, and are instead in $\mathrm{SL}(d;\mathbb{R})$.

Hence we combine the K estimates for $\varphi^{-1}(p)$ with weights w_i to give

$$q = \frac{\sum_i w_i q_i}{\sum_i w_i}$$

and we interpolate the deformed transformation matrix in the same way as

$$Q = \frac{\sum_i w_i Q'_i Q_i^{-1}}{\sum_i w_i}$$

Here the function $\textsc{ClosestPoints}(x, \{X_i\})$ returns k indices of up to K

Algorithm 13 Deform with point cloud

1: **procedure** DEFORMPOINTCLOUD
Input: Deform Particles $\alpha(t) = \{\alpha_1(t), \ldots, \alpha_n(t)\}$ for t in $t_1, \ldots, t_N$
Input: Position $q \in \mathbb{R}^d$ at time t_1
Output: Deformed position $q \in \mathbb{R}^d$ at time t_N
Output: Deformation gradient $Q \in \mathrm{SL}(d;\mathbb{R})$
2: $I = \{j_1, \ldots, j_k\} \leftarrow \textsc{ClosestPoints}(x, \alpha(t_1))$
3: $(X_i, Q_i) \leftarrow \alpha_{j_i}(t_1)$
4: $w_i \leftarrow \textsc{Weight}(p - X_i)$
5: $(X_i', Q_i') \leftarrow \alpha_{j_i}(t_N)$
6: $q_i \leftarrow Q_i' Q_i^{-1}(p - X_i) + X_i'$
7: $q \leftarrow \sum w_i q_i / \sum w_i$
8: $Q \leftarrow \sum w_i Q_i' Q_i^{-1} / \sum w_i$

closest points to x. The weights are calculated with the WEIGHT function $w(x) = 1/x^2$, although different functions can be used to tune the resulting deformation for example $w(x) = x^{-\gamma}$ for $\gamma > 1$.

7.6 Conclusions

In this chapter we have presented the algorithms we developed in the order they were required during this thesis.

The forward deformation technique requires a triangle look up, and is only piecewise linear for density and piecewise constant for velocity interpolation. However it was useful for the initial investigations and proof of concept.

The backwards deformation technique allows a simpler interpolation of data on the initial regular grid, and allows us to find pre-images for exactly the points we need in our deformed space. However the large data sets in today's productions make it prohibitive to evaluate the jet particle flow on each point of interest.

Calculating the deformation instead on a coarse grid allows the greater efficiency. The most intensive part of the calculation, namely evaluating the Hamiltonian flow of each point, can be reduced significantly. A reduction of the grid resolution by a factor of k reduces the number of points requiring simulation by k^d for dimension d.

Taking advantage of the fact that we know the deformation gradient of the flow

at each deformed point as a by product of integrating the flow, we can use this to calculate a bezier volume for the deformation. This takes full advantage of all the data we have calculated and allows more accurate interpolation.

Finally, by removing the requirement for calculating the deformed points on a regular grid and using an arbitrary point cloud to store the data, we can reuse the deformation data for any frame. The jet particle flow can be calculated once and stored off-line, after which the deformation between arbitrary frames can be calculated. This improves efficiently when calculating the deformation on a sequence of frames sharing the same flow.

Without these techniques it would be prohibitively expensive to deform the large data sets used in production today.

See the tools Section A.3.1 for the details of how we implement these algorithms using the OpenVDB [104] toolkit. The implementation leverages and maintains the sparsity of the volume data which the OpenVDB library supports. The ability to work directly with this toolkit allows us to integrate these methods easily with current visual effects applications including SideFX HoudiniFX and Autodesk Maya.

We are now able to create arbitrary deformations with jet particle flow which we use particularly in Chapters 4 and 5.

Chapter 8

Conclusions

8.1 Summary

Here we summarise the main contributions of this thesis outlined in Chapter 1.

1. To support Hypothesis 1 we have demonstrated a method for sketching realistic, incompressible and fluid-like flows in real-time using jet particles as a natural interface for the artist.

   Using jet particle dynamics with a sparse number of particles, we can create complex flows whose divergence-free velocity fields can be evaluated at arbitrary resolutions. The velocity field produced is divergence-free and can be evaluated efficiently without a projection step to enforce this condition.

   Unlike the shape sculpting system of Von Funck, Theisel, and Seidel [29], our edits are naturally fluid-like due to their links to Euler's equations.

   We have demonstrated instabilities observed in the dynamics of interacting jet particles. To counter these, we have shown that the use of velocity constraints can be effective in delaying their onset. We suggest further avenues in order to resolve these issues below.

2. We have provided methods for deforming flows using jet particles in support of Hypothesis 2. To this end, we have created a system for designing new variations of existing flows in a production environment.

   By providing a method of coupling the jet particles with a given flow, we show

how to create a local temporal edit to the fluid after which the fluid is allowed to return its original configuration. In contrast to the liquid editing system proposed by Pan et al. [52], our edits are incompressible without requiring an optimisation step.

We have demonstrated fluid editing using smoke simulations in 3D from a production environment, and we implement a solution for deforming fluid caches in Maya using curve rigs to render new variations of the flow.

3. To support Hypothesis 3 we have demonstrated a method of using jet particles to represent existing vector fields, and have shown how to design new ones using constraints on the particle momenta. To achieve this aim we show a method for solving the momenta of the jet particles given their velocity field and the velocity gradient at the particle locations.

   Using this method we show how to design vector fields that follow a given curve, and we use the extra degrees of freedom of the 1-jet particles to constrain the deformation gradient around the curve to control the twisting of the vector field's flow.

   The method does not require the points to lie on a curve however, and the jet particles state can be used to edit the divergence-free vector field directly. In this way the jet particles provide a more intuitive way to create divergence-free vector fields than using curl-noise [37], since the input for that system is a scalar field and it is not easy to control the direction of the vector field directly.

4. In order to demonstrate the techniques in our sketching and editing applications for contributions (1-2) we have provided implementations for deforming functions, densities, vector fields and tensors. This allows us to use the same framework to transform meshes, fluid volumes, level sets and rigid bodies. We have provided direct methods for calculating deformations and approximation methods for allowing the deformation to be evaluated efficiently on sparse sets of points for use on large production data sets. These methods

take advantage of a moving frame carried by the jet particle flow that shears with the flow. Even large deformations are well represented without loss of detail by the algorithms since the flow is Hamiltonian, and therefore energy preserving, reversible and presents no numerical viscosity.

We think that there will be further uses of jet particles in computer graphics and we outline ideas for future work in the next section.

8.2 Future Work

Ideas for future work are outlined below.

8.2.1 Deformation

A related work in shape deformation [29] constructs divergence-free vector fields in 3D by taking the cross product of the gradient of two scalar functions $\nabla e \times \nabla f$. Solomon et al. [105] propose a method for sketching near-isometric deformations by recording a brush stroke to accumulate a deformation, in similar way to our fluid sketching application in Chapter 4.

Hence our work can have applications in shape modelling, since our edits have the same desired properties: they are smooth, volume preserving, they do not generate self-intersections and they preserve high frequencies. Moreover the sketching methods we have designed will work with surfaces defined by level sets, point cloud data, meshes and volumes alike.

8.2.2 Jet Particle Simulations

We have demonstrated that we can use jet particles to model fluid-like flow, and we have shown how to constrain the particles to increase stability and control the flow.

Further, jet particle simulations do not require a velocity projection step. For other grid based fluid solvers, just this step alone can cost around one quarter of the simulation time [106]. It would be fascinating to evaluate how suitable jet particles are for fluid simulation in their own right, and whether they can be combined with an SPH solver effectively.

We have seen that jet particle dynamics show instabilities and do not approxi-

mate Eulerian flow well when they are not evenly spaced. It is an open question as to whether we can stabilise jet particles simulations, but one possible line is to try the re-positioning of the particles when they get too close, using a momenta projection step as described in Chapter 4. The particles could be reset back to a regular grid as in the particle mesh method of Cotter, Frank, and Reich [107]. Alternatively they may be allowed to move dynamically with policies in place to keep them well distributed. Possible methods include moving the particles to the center of their power diagrams as with the power particles method [92], or adding separation forces to keep the particles separate to prevent blow up.

Moreover it may be possible to increase the efficiency of the of the pair-wise evaluation of the matrix kernels by using FMM (Fast Multipole Method), a method used by Angelidis [108] to create a multi-scale Lagrangian smoke solver. The FMM approach has also been applied successfully in the multi-scale LDDKBM framework [60] to accelerate the particle flows and to parallelise the problem with a GPU implementation.

It may also be possible to include viscosity in the particle interaction as done in SPH, by considering the Lagrangian formulation of the Navier-Stokes equations. Another possibility is to use an interpolated matrix kernel to relax the divergence-free condition (as discussed in Section 3.4) to model diffusion.

Jet particles may also have a significant role to play in adding turbulent detail at finer scales to fluid simulations.

We may also seek to couple jet particle flow with another jet particle simulation at a different resolution. We discussed the advection of jet particles to model turbulence within a flow in Chapter 5, and this is an example of the coupling of one flow with another dynamically.

Holm and Tronci [109] create a coupled model for fluid flow, where one fluid flow is the mean flow for dynamics at a finer scale. This approach may be useful as it allows simulations with different sized particles to interact. This is not possible within a jet particle simulation as it breaks the particle relabelling symmetry. Hence we may consider a multi-level jet particle simulation where each level is a constraint

or background velocity field for the next, and so the conservation of momentum is still upheld within each level. This model can provide two way interaction between levels, modelling both back-scattering and forwards-scattering.

Appendix A

Tools

In the sections that follow we review the tools that we have created and used in the previous chapters.

A.1 Manta

We used manta [91] as a basis for our fluid paint application shown in Video 4.4 in Chapter 4. To make this possible we extended manta to allow the sketching of deformations interactively with 1-jet particles. Manta is also used to generate the 2D smoke simulations shown in Chapter 5.

A.2 Tensor Library

When developing the sketching application from Chapter 4 and the editing applications described in Chapter 5, we needed to evaluate the jet particle's velocity fields and simulate their interaction.

Calculating the velocities and solving the equations of motion for the jet particles requires evaluation of the matrix kernel and their derivatives. This in turn calls for a high order tensor maths library.

When programming in scientific python, the arbitrary dimensional tensor library *einsum* is available. However, evaluation in C++ is required for the integration into SideFX Houdini FX and Autodesk Maya 2016, and for our custom sketching application. In addition to evaluating the velocity fields, fifth order tensors are required for evaluating the equations of motion for 1-jet particles in Equations (3.10),

(3.11) and (3.12).

We evaluated the C++ tensor library Einsum [110], but it did not have the necessary performance, and its requirements for tracking tensor indices made it prohibitive to use. However, the solution we generated for tensor evaluation was validated against Einsum as a reference implementation.

The most interesting alternative LTensor [111] is a tensor library written in C++ that uses templates. Operations are reduced in-line by the compiler and therefore it has much better runtime performance, although the use of the higher order tensors requires an increase in compile time.

However LTensor only supports operations on tensors up to order four. The author confirmed that no code was available to extend the library source code, so we extended the library using custom python code. This generated the C++ template functions we needed for the fifth order tensor calculations we required.

A.3 OpenVDB

In Chapter 7 we described the algorithm for evaluating the deformation given by jet particle flow, including a method for pushing forward velocity vectors. In this Section we describe the implementation of these methods for use with OpenVDB grids.

OpenVDB [104] is an open source C++ library for storing volume data in three dimensions. It uses sparse hierarchical structures for efficiency and also provides useful tools for data manipulation. The Tree data structure is responsible for storing data at each (i, j, k) index of space, where each index is a 32-bit integer. By default the data structure uses Leaf nodes to store an $8 \times 8 \times 8$ voxel grid. Each OpenVDB file may contain multiple grids, each with its own Tree structure.

Figure A.1a shows a visualisation of the leaf nodes for a smoke density grid from an OpenVDB file. In designing our deformation algorithm, we take care to minimise the deformation calculations and storage requirements. We do this by taking the Tree data structure into account for each field stored in the OpenVDB file. We outline the algorithms in the sections that follow.

A.3.1 Deformation of OpenVDB Files

In order to keep the deformed data structures sparse, and to reduce computation in the case when we have many overlapping grids, we proceed as follows.

Firstly we create a sparse region in space that contains the image of the deformation of the source grid. This region can be strictly larger and coarser than the exact image, since it will be used to create an empty *deform grid* Φ as outlined in Section 7.3. Secondly we fill the *deform grid* Φ with the position deltas and matrices describing the deformation. Finally we evaluate the deformed grids at each potential point in the deformed region by applying Φ to each of the input grids.

Algorithm 14 VDB Deform Region

Input: Grid to deform G
Input: Deform grid voxel size dx
Output: Deform grid Φ

```
1:  procedure DEFORMREGION
2:      for i in NUMLEAFNODES(G) do:
3:          L ← Leaf node i of G
4:          p_j ← Bounding box vertices of L
5:          for j in 1,...,2^d do:
6:              p_j ← FLOWFORWARDS(p_j)
7:          B_i ←Bounding box of p_j
8:      Φ ←Deform grid CREATEGRID(∪_i B_i, dx)
9:      for all B_i do
10:         for all cells c in B_i do
11:             ACTIVATECELL(Φ, c)
```

In DEFORMREGION in Algorithm 14 we created a deform grid with voxel width dx. Cells are tagged as active with the function ACTIVATECELL(Φ, c) if they are an image of any leaf bounding box region of a source grid under the jet particle flow.

Here the function NUMLEAFNODES(G) returns the number of leaf nodes in the grid G, and CREATEGRID(B, dx) creates a grid with voxel size dx and a transform based on the bounding box of the region B. Usually we choose a voxel size around five times larger than the grid in order to reduce computation. This results in very little loss of visual quality.

This method requires evaluating the FLOWFORWARDS function on each vertex

of the leaf bounding box vertex, this set being significantly smaller than the number of voxels in the source grid. However if the number of leaf box vertices becomes significant we can still optimise this process by creating a coarse grid for the forward deformation as outlined in Section 7.3. In our implementation we use only the leaf bounding box vertices.

Algorithm 15 Create Deform Grid

Input: Empty deform grid Φ
Input: Jet Particles $J(t) = \{j_1(t), \ldots, j_n(t)\}$ for t in $t_1, \ldots, t_N$
Input: Deform grid Φ

```
1: procedure FILLDEFORMGRID
2:     for all Active cell c in grid Φ do
3:         p ← Center of cell c
4:         (x', Q) ← FLOWBACKWARDS(J(t), x)
5:         Φ[c] ← (x' − x, Q)
```

In FILLDEFORMGRID from Algorithm 15, we calculate the deltas dp and the deformation frame Q based on the backwards flow of the jet particles. These are calculated for each active cell and stored as vectors in the deform grid Φ. See Figure A.2 for an image showing the delta vectors calculated from the sparse smoke grid shown in Figure A.1a, with a deformation defined by a stationery 1-jet particle representing a rotation.

Algorithm 16 Create Deformed Grid

Input: Source grid G
Output: Deformed grid H

```
1: function DEFORMGRID(G)
2:     for all Deformed leaf Bounding box B_i of grid G do
3:         for all cells c in B_i do
4:             ACTIVATECELL(H, c)
5:     for all Active cell c in grid H do
6:         p ← Center of cell c
7:         (dp, Q') ← SAMPLEDEFORMGRID(Φ, p)
8:         d, u' ← SAMPLEGRID(G, p)
9:         u ← Q' · u'
10:        H[c, D] ← d
11:        H[c, U] ← u
12:    return H
```

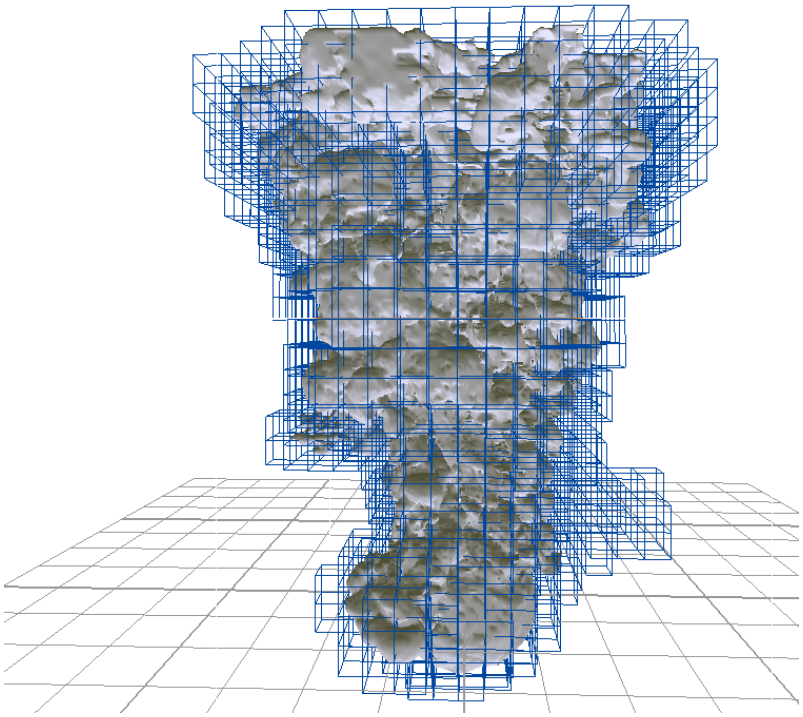
Finally we apply the deformation stored in the deform grid Φ to each source

grid in the OpenVDB file.

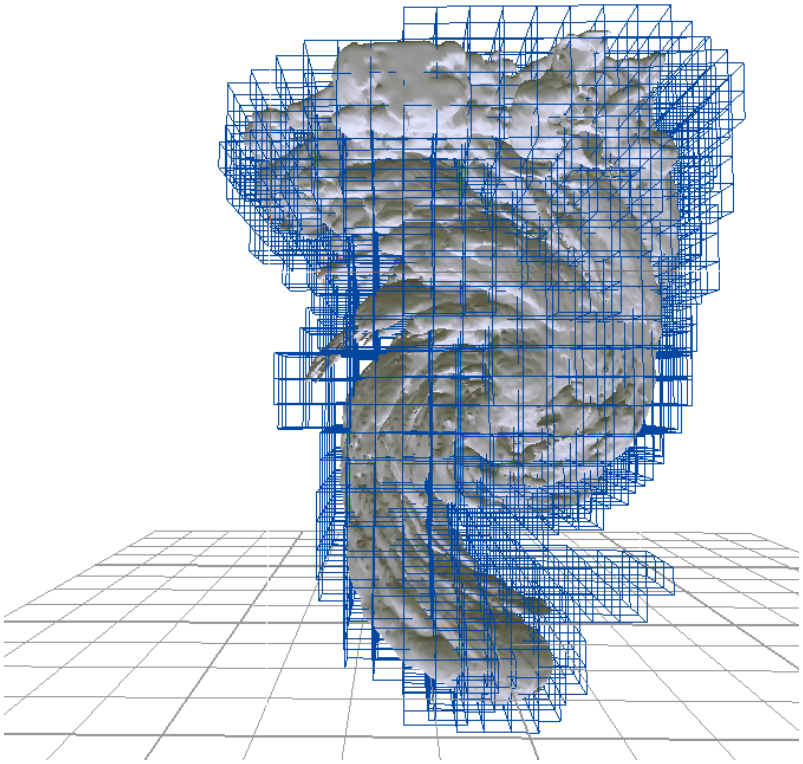
Using Algorithm 16, we find the set of cells that could be the image of cells in the source grid under the flow, and trace each position back using the deform grid Φ. This uses the function SAMPLEDEFORMGRID(Φ, p) which will return an interpolated position delta and deformation frame, based on the trilinear or bezier volume methods defined in Sections 7.3 and 7.4 respectively.

Figure A.1b shows a deformed version of the smoke density field from Figure A.1a under the action of a 1-jet particle flow, calculated by applying the Φ shown in A.2. Note how the leaf nodes have only been created where required to maintain the sparse representation.

As a side, the functions defined above have been implemented to take advantage of the multithreading using TBB [112]. Since the set of active voxels are available, the OpenVDB framework will call our functions only with these voxels, having split the work into blocks for multiple threads.



(a) Leaf nodes for a smoke density field



(b) Leaf nodes for the deformed density field

Figure A.1: The deformation algorithm preserves the sparsity of the density field

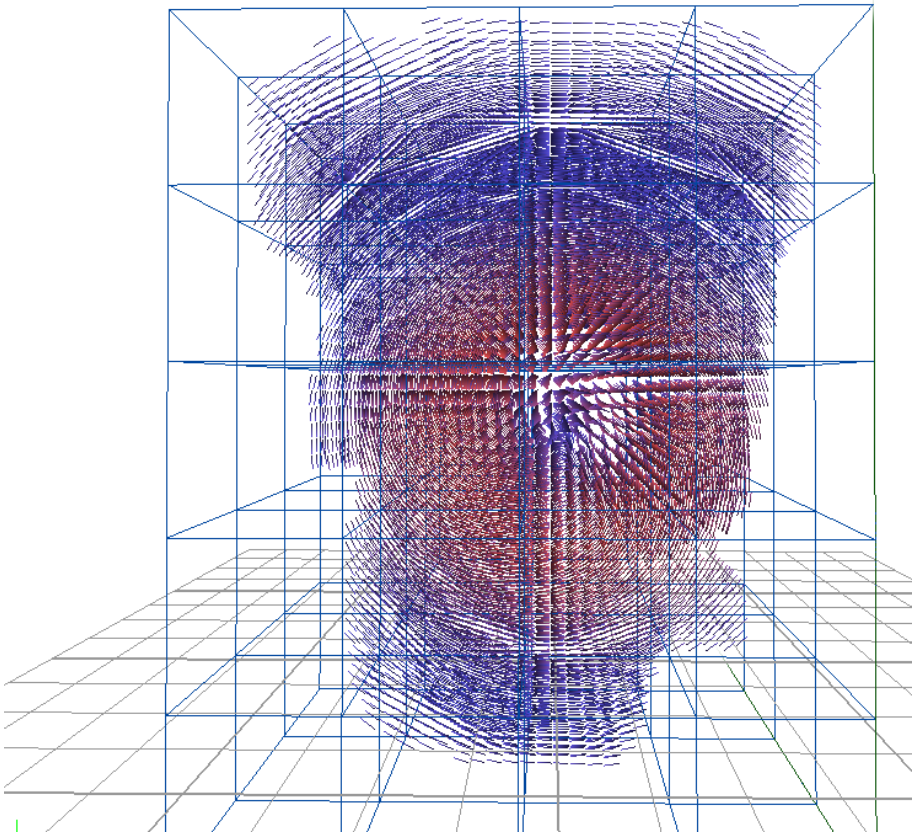


Figure A.2: The delta vectors used for the deformation. These are defined sparsely on a coarser grid than the data set being deformed.

A.3.2 Visualisation in Maya

In order to implement the Maya rigs described in Section 5.6, we implemented a ray traced GLSL visualisation of the OpenVDB grids. The OpenVDB source code provided a visualiser in Maya for OpenVDB files, but it did not use ray tracing and so the results were hard to read.

In order to perform the fast evaluation of the volume deformation on the GPU, the velocity field from the 1-jet particles must be evaluated. This requires calculating the first derivative of the matrix kernel and the contraction with μ, which is a third order tensor operation. Although GLSL only supports matrix operations natively, the use of a matrix array structure aids the writing of tensor arithmetic functions.

Listing A.1 shows an example of GLSL code creating a third order tensor using such a data structure.

Listing A.1: Momentum Matrix

```
struct ten3 {
  mat3 a[3];
};

ten3 mij_vk(mat3 m, vec3 v)
{
  ten3 r;
  for (int k=0; k<3; k++) {
      for (int j=0; j<3; j++) {
          for (int i=0; i<3; i++) {
              r.a[i][j][k] = m[i][j]*v[k];
          }
      }
  }
  return r;
}
```

This allows the 1-jet velocity field to be calculated without extra complexity.

A.3.3 Deformation in Maya Rig

The steps required for implementing the deformer rig in Maya described in Section 5.6 are presented in this section.

The purpose of these tools is to allow a curve to be used to deform an openVDB cache interactively with a low resolution proxy, since the production data set is very large. Finally the deformation is calculated on the full resolution volume in software

for rendering.

Firstly, OpenVDB proxy files are automatically computed as a post-process after the fluid simulation is complete. Large grids are reduced to files no larger than $30 \times 30 \times 30$ voxels so that it is still possible to visualise the files interactively over the network.

Then the jet particle flow is loaded into memory, representing the edit flow. The flow may be provided from the description of curves in the maya scene, or from a network file. The jet particles are stored efficiently as a particle cache carrying extra information. In particular we store an extra vector p, a matrix μ and a radius σ per frame.

In order to carry out the deformation, we may use the C++ path described in Section A.3.1. The GPU path is similar, except it is applied to proxy volume data loaded as a 3D texture. The deformation is calculated using a compute shader in GLSL that evaluates the velocity field sequentially, and integrates the jet particle positions.

Finally to facilitate the rendering with Arnold, we wrote a volume translator which deforms the OpenVDB file in software at render time.

A.3.4 Calculating the Projection Matrix

Here we show the C++ code used for calculating the block matrices M from Section 6.1.

We define the matrix eA and vector eB corresponding to the matrix M, and the arrays v and dv representing the observed velocity data and gradient. We represent the data structures using the Eigen C++ package [113].

A.3.4.1 Projection matrix for 0-Jet Particles

The code in Listing A.2 shows the algorithm for creating the matrix M from Section 6.1.2 used for projecting 0-jet particles.

Listing A.2: Momentum matrix for 0-jets

```
// Fill Matrix
for ( int a=0; a<n; a++) {
    for ( int b=0; b<n; b++) {
        for ( int i=0; i<d; i++) {
```

```
            for (int j=0; j<d; j++) {
                // K{ij} p{k} [UL]
                eA( a*d + i, b*d + j) = K(i, j);
            }
        }
        // u
        for ( int a=0; a<n; a++) {
            const Vector<d> &vel = v[a];
            for ( int i=0; i<d; i++) {
                eb(a*d + i) = vel.data()[i];
            }
        }
    }
}
```

A.3.4.2 Projection matrix for 1-Jet Particles

The code in Listing A.3 shows the algorithm for creating the matrix M from Section 6.1.3 used for projecting 1-jet particles, including the application of the incompressibility constraint.

Listing A.3: Momentum matrix for 1-jets

```
// Fill Matrix
for ( int a=0; a<n; a++) {
    for ( int b=0; b<n; b++) {
        for ( int i=0; i<d; i++) {
            for (int j=0; j<d; j++) {
                // K{ij} p{k} [Upper Left]
                eA( a*d + i, b*d + j) = K(i, j);

                for (int k=0; k<d; k++) {
                    // d{k}K{ij} mu{jk} [Upper Right]
                    eA( a*d + i, n*d + b*d*d + j*d + k) = -DK(i, j, k);

                    // dv{i}d{j} = d{j}K{ik} p{k} [Bottom Left]
                    eA( n*d + a*d*d + i*d + j, b*d + k) = DK(i, k, j);

                    for (int l = 0; l < d; l++) {
                        // d{j}u{i} = -d{kj}K{il} mu{lk} [Bottom Right]
                        eA(n*d + a*d*d + i*d + j, n*d + b*d*d + l*d + k) = -D2K(i, l, k, j);
                    }
                }
            }
        }
    }
}
// Incompressibility Condition
for (int a=0; a<n; a++) {
    // update row for the last component of mu_a
    for (int b=0; b<d*n + d*d*n; b++) {
        // zero the whole d*d-1 row
        eA(n*d + a*d*d + (d-1)*d + d-1, b) = 0;
    }
    for (int b=0; b<d; b++) {
        // update each mu_bb coefficient
        eA(n*d + a*d*d + (d-1)*d + d-1, n*d + a*d*d + b*d + b) = 1.;
    }
```

```
}
// u and du
for ( int a=0; a<n; a++) {
    const Vector<d> &vel = v[a];
    const Matrix<d> &dvdx = dv[a];
    for ( int i=0; i<d; i++) {
        eb(a*d + i) = vel.data()[i];
        for (int j = 0; j < d; j++) {
            eb(n*d + a*d*d + i*d + j) = dvdx(i,j);
        }
    }
    // Incompressibility Condition for mu_a (RHS)
    eb(n*d + a*d*d + (d-1)*d + d-1) = 0.;
}
```

A.3.5 Houdini Implementation

We use an *Attribute Wrangle SOP* node to evaluate the velocity field and its gradient on the points. The code, which uses the custom *VEX* matrixKernel function, is described in Listing A.4 below. It collects the summed contributions from all of jet particles at each point.

Listing A.4: Velocity and gradient evaluation

```
vector  dq, v    = {0,0,0};
matrix3 dv, dvdx = {0,0,0,0,0,0,0,0,0};

int handle = pcopen(@OpInput2, "P", @P, max_radius, max_points);
while(pciterate(handle))
{
    pcimport(handle, "P", p2);
    pcimport(handle, "mom", mom2);
    pcimport(handle, "mu", mu2);
    matrixKernel(@P, p2, mom2, mu2, sigma, dq, dv);
    v += dq;
    dvdx += dv;
}

v@v = v;
3@dvdx = dvdx;
```

In order to invert the matrix we implemented a custom *Solve Momenta SOP* which builds the block matrices and inverts the linear system of equations using Listing A.3.4.2 as described above in Section 6.3.3.

Bibliography

[1] N. Foster and D. Metaxas. “Realistic animation of liquids”. In: *Graphical models and image processing* 58.5 (1996), pp. 471–483.

[2] J. Stam. “Stable fluids”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1999, pp. 121–128.

[3] J. J. Monaghan. “Smoothed particle hydrodynamics”. In: *Annual review of astronomy and astrophysics* 30 (1992), pp. 543–574.

[4] Y. Zhu and R. Bridson. “Animating sand as a fluid”. In: *ACM Transactions on Graphics (TOG)*. Vol. 24. 3. ACM. 2005, pp. 965–972.

[5] M. Lentine, J. T. Grétarsson, and R. Fedkiw. “An unconditionally stable fully conservative semi-Lagrangian method”. In: *Journal of computational physics* 230.8 (2011), pp. 2857–2879.

[6] J. Stam. “Flows on surfaces of arbitrary topology”. In: *ACM Transactions On Graphics (TOG)*. Vol. 22. 3. ACM. 2003, pp. 724–731.

[7] S. Elcott, Y. Tong, E. Kanso, P. Schröder, and M. Desbrun. “Stable, circulation-preserving, simplicial fluids”. In: *ACM Transactions on Graphics (TOG)* 26.1 (2007), p. 4.

[8] P. Mullen, A. McKenzie, D. Pavlov, L. Durant, Y. Tong, E. Kanso, J. E. Marsden, and M. Desbrun. “Discrete Lie advection of differential forms”. In: *Foundations of Computational Mathematics* 11.2 (2011), pp. 131–149.

[9] T. De Witt, C. Lessig, and E. Fiume. “Fluid simulation using laplacian eigenfunctions”. In: *ACM Transactions on Graphics (TOG)* 31.1 (2012), p. 10.

[10] O. Azencot, S. Weißmann, M. Ovsjanikov, M. Wardetzky, and M. Ben-Chen. “Functional fluids on surfaces”. In: *Computer Graphics Forum*. Vol. 33. 5. Wiley Online Library. 2014, pp. 237–246.

[11] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner. “SPH fluids in computer graphics”. In: (2014).

[12] M. Desbrun and M.-P. Gascuel. *Smoothed particles: A new paradigm for animating highly deformable bodies*. Springer, 1996.

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

[14] I. Alduán and M. A. Otaduy. “SPH granular flow with friction and cohesion”. In: *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*. ACM. 2011, pp. 25–32.

[15] B. Ren, C. Li, X. Yan, M. C. Lin, J. Bonet, and S.-M. Hu. “Multiple-fluid SPH simulation using a mixture model”. In: *ACM Transactions on Graphics (TOG)* 33.5 (2014), p. 171.

[16] X. Yan, Y.-T. Jiang, C.-F. Li, R. R. Martin, and S.-M. Hu. “Multiphase SPH simulation for interactive fluids and solids”. In: *ACM Transactions on Graphics (TOG)* 35.4 (2016), p. 79.

[17] J. Cornelis, M. Ihmsen, A. Peer, and M. Teschner. “Liquid boundaries for implicit incompressible SPH”. In: *Computers & Graphics* 52 (2015), pp. 72–78.

[18] B. Ren, X. Yan, T. Yang, C.-f. Li, M. C. Lin, and S.-m. Hu. “Fast SPH simulation for gaseous fluids”. In: *The Visual Computer* 32.4 (2016), pp. 523–534.

[19] R. Winchenbach, H. Hochstetter, and A. Kolb. “Infinite continuous adaptivity for incompressible SPH”. In: vol. 36. 4. ACM, 2017, p. 102.

[20] M. B. Liu and G. R. Liu. “Smoothed Particle Hydrodynamics (SPH): an Overview andRecent Developments”. In: *Archives of Computational Methods in Engineering* 17.1 (2010), pp. 25–76. ISSN: 1886-1784.

[21] J. Brackbill and H. Ruppel. “FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions”. In: *Journal of Computational Physics* 65.2 (1986), pp. 314–343.

[22] K. Raveendran, N. Thuerey, C. Wojtan, and G. Turk. “Controlling liquids using meshes”. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 2012, pp. 255–264.

[23] F. Ferstl, R. Ando, C. Wojtan, R. Westermann, and N. Thuerey. “Narrow band FLIP for liquid simulations”. In: *Computer Graphics Forum*. Vol. 35. 2. Wiley Online Library. 2016, pp. 225–232.

[24] J. Wejchert and D. Haumann. “Animation aerodynamics”. In: *ACM SIGGRAPH Computer Graphics*. Vol. 25. 4. ACM. 1991, pp. 19–22.

[25] M. B. Nielsen and B. B. Christensen. “Improved variational guiding of smoke animations”. In: *Computer Graphics Forum*. Vol. 29. 2. Wiley Online Library. 2010, pp. 705–712.

[26] Z. Yuan, F. Chen, and Y. Zhao. “Pattern-guided smoke animation with lagrangian coherent structure”. In: *ACM transactions on graphics (TOG)*. Vol. 30. 6. ACM. 2011, p. 136.

[27] M. B. Nielsen and R. Bridson. “Guide shapes for high resolution naturalistic liquid simulation”. In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 83.

[28] A. Stomakhin and A. Selle. “Fluxed animated boundary method”. In: vol. 36. 4. ACM, 2017, p. 68.

[29] W. Von Funck, H. Theisel, and H.-P. Seidel. "Vector field based shape deformations". In: *ACM Transactions on Graphics (TOG)* 25.3 (2006), pp. 1118–1125.

[30] A. Treuille, A. Lewis, and Z. Popović. "Model reduction for real-time fluids". In: *ACM Transactions on Graphics (TOG)*. Vol. 25. 3. ACM. 2006, pp. 826–834.

[31] B. Liu, G. Mason, J. Hodgson, Y. Tong, and M. Desbrun. "Model-reduced variational fluid simulation". In: *ACM Transactions on Graphics (TOG)* 34.6 (2015), p. 244.

[32] M. Wicke, M. Stanton, and A. Treuille. "Modular bases for fluid dynamics". In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 39.

[33] T. Kim and J. Delaney. "Subspace Fluid Re-simulation". In: *ACM Trans. Graph.* 32.4 (July 2013), 62:1–62:9. ISSN: 0730-0301. DOI: 10.1145/2461912.2461987. URL: http://doi.acm.org/10.1145/2461912.2461987.

[34] S. Weißmann, U. Pinkall, and P. Schröder. "Smoke rings from smoke". In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 140.

[35] S. Eberhardt, S. Weissmann, U. Pinkall, and N. Thuerey. "Hierarchical vorticity skeletons". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM. 2017, p. 6.

[36] D Mumford and P. W. Michor. "On Euler's equation and 'EPDiff'". In: *Journal of Geometric Mechanics* 5.3 (2013). arXiv:1209.6576 [math.AP], pp. 319–344.

[37] R. Bridson, J. Houriham, and M. Nordenstam. "Curl-noise for procedural fluid flow". In: *ACM Transactions on Graphics (TOG)* 26.3 (2007), p. 46.

[38] K. Perlin. "An image synthesizer". In: *ACM Siggraph Computer Graphics* 19.3 (1985), pp. 287–296.

[39] J. Steinhoff and D. Underhill. “Modification of the Euler equations for vorticity confinement: Application to the computation of interacting vortex rings”. In: *Physics of Fluids* 6.8 (1994), pp. 2738–2744.

[40] A. Selle, N. Rasmussen, and R. Fedkiw. “A vortex particle method for smoke, water and explosions”. In: *ACM Transactions on Graphics (TOG)*. Vol. 24. 3. ACM. 2005, pp. 910–914.

[41] T. Kim, N. Thürey, D. James, and M. Gross. “Wavelet turbulence for fluid simulation”. In: *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, p. 50.

[42] A. Tsinober. “Turbulence: The Legacy of A. N. Kolmogorov. By U. Frisch. Cambridge University Press, 1995. 296 pp. ISBN 0 521 45713 0. 15.95.” In: *Journal of Fluid Mechanics* 317 (1996), 407410. DOI: 10.1017/S0022112096210791.

[43] T. Pfaff, N. Thuerey, J. Cohen, S. Tariq, and M. Gross. “Scalable fluid simulation using anisotropic turbulence particles”. In: *ACM Transactions on Graphics (TOG)* 29.6 (2010), p. 174.

[44] B. Launder and B. Sharma. “Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc”. In: *Letters in Heat and Mass Transfer* 1.2 (1974), pp. 131 –137. ISSN: 0094-4548. DOI: http://dx.doi.org/10.1016/0094-4548(74)90150-7. URL: http://www.sciencedirect.com/science/article/pii/0094454874901507.

[45] T. Kim, J. Tessendorf, and N. Thuerey. “Closest point turbulence for liquid surfaces”. In: *ACM Transactions on Graphics (TOG)* 32.2 (2013), p. 15.

[46] O. Mercier, C. Beauchemin, N. Thuerey, T. Kim, and D. Nowrouzezahrai. “Surface turbulence for particle-based liquid simulations”. In: *ACM Transactions on Graphics (TOG)* 34.6 (2015), p. 202.

[47] F. Pighin, J. M. Cohen, and M. Shah. “Modeling and editing flows using advected radial basis functions”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2004, pp. 223–232.

[48] A. Treuille, A. McNamara, Z. Popović, and J. Stam. “Keyframe control of smoke simulations”. In: *ACM Transactions on Graphics (TOG)*. Vol. 22. 3. ACM. 2003, pp. 716–723.

[49] A. McNamara, A. Treuille, Z. Popović, and J. Stam. “Fluid control using the adjoint method”. In: *ACM Transactions On Graphics (TOG)*. Vol. 23. 3. ACM. 2004, pp. 449–456.

[50] L. Shi and Y. Yu. “Taming liquids for rapidly changing targets”. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM. 2005, pp. 229–236.

[51] S. Zhang, X. Yang, Z. Wu, and H. Liu. “Position-based fluid control”. In: *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. ACM. 2015, pp. 61–68.

[52] Z. Pan, J. Huang, Y. Tong, C. Zheng, and H. Bao. “Interactive localized liquid motion editing”. In: *ACM Transactions on Graphics (TOG)* 32.6 (2013), p. 184.

[53] K. Raveendran, C. Wojtan, N. Thuerey, and G. Turk. “Blending liquids”. In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 137.

[54] N. Thuerey. “Interpolations of smoke and liquid simulations”. In: *ACM Transactions on Graphics (TOG)* 36.1 (2016), p. 3.

[55] Z. Pan and D. Manocha. “Efficient Optimal Control of Smoke using Space-time Multigrid”. In: *arXiv preprint arXiv:1608.01102* (2016).

[56] P.-L. Manteaux, U. Vimont, C. Wojtan, D. Rohmer, and M.-P. Cani. “Space-time sculpting of liquid animation”. In: *Proceedings of the 9th International Conference on Motion in Games*. ACM. 2016, pp. 61–71.

[57] M. Chu and N. Thuerey. “Data-driven synthesis of smoke flows with CNN-based feature descriptors”. In: *arXiv preprint arXiv:1705.01425* (2017).

[58] L. Younes. *Shapes and diffeomorphisms*. Vol. 171. Springer Science & Business Media, 2010.

[59] S. Sommer, M. Nielsen, F. Lauze, and X. Pennec. “A Multi-Scale kernel bundle for LDDMM: towards sparse deformation description across space and scales”. In: *Information Processing in Medical Imaging*. Springer. 2011, pp. 624–635.

[60] S. Sommer. “Accelerating multi-scale flows for LDDKBM diffeomorphic registration”. In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 499–505. DOI: 10.1109/ICCVW.2011.6130284.

[61] S. C. Joshi and M. I. Miller. “Landmark matching via large deformation diffeomorphisms”. In: *Image Processing, IEEE Transactions on* 9.8 (2000), pp. 1357–1370.

[62] M. Micheli and J. Glauns. “Matrix-valued Kernels for Shape Deformation Analysis”. In: *IEEE Transactions on Visualization and Computer Graphics* (2013), pp. 1–48.

[63] M. Bauer, M. Bruveris, and P. W. Michor. “Overview of the geometries of shape spaces and diffeomorphism groups”. In: *Journal of Mathematical Imaging and Vision* 50.1-2 (2014), pp. 60–97.

[64] V. I. Arnold. “Sur la géométrie différentielle des groupes de Lie de dimension infinie et ses applications à l’hydrodynamique des fluides parfaits”. In: *Annales de l’Institute Fourier* 16 (1966), pp. 316–361.

[65] D. Pavlov, P. Mullen, Y. Tong, E. Kanso, J. E. Marsden, and M. Desbrun. “Structure-preserving discretization of incompressible fluids”. In: *Physica D: Nonlinear Phenomena* 240.6 (2011), pp. 443–458.

[66] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun. "Energy-preserving integrators for fluid animation". In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 38.

[67] H. O. Jacobs, T. S. Ratiu, and M. Desbrun. "On the coupling between an ideal fluid and immersed particles". In: *Physica D: Nonlinear Phenomena* 265 (2013), pp. 40–56.

[68] C. Cotter, D. Holm, H. Jacobs, and D. Meier. "A jetlet hierarchy for ideal fluid dynamics". In: *Journal of Physics A: Mathematical and Theoretical* 47.35 (2014), p. 352001.

[69] C. J. Cotter, J. Eldering, D. D. Holm, H. O. Jacobs, and D. M. Meier. "Weak dual pairs and jetlet methods for ideal incompressible fluid models in $n \geq 2$ dimensions". In: *Journal of Nonlinear Science* 26.6 (2016), pp. 1723–1765.

[70] D. D. Holm, T. Schmah, C. Stoica, and D. C. Ellis. *Geometric mechanics and symmetry: from finite to infinite dimensions*. 12. Oxford University Press London, 2009.

[71] S. Sommer and H. O. Jacobs. "Reduction by Lie Group Symmetries in Diffeomorphic Image Registration and Deformation Modelling". In: *Symmetry* 7.2 (2015), pp. 599–624.

[72] H. O. Jacobs and S. Sommer. "Higher-order spatial accuracy in diffeomorphic image registration". In: *arXiv preprint arXiv:1412.7504* (2014).

[73] M. I. Miller, A. Trouvé, and L. Younes. "On the metrics and Euler-Lagrange equations of computational anatomy". In: *Annual review of biomedical engineering* 4.1 (2002), pp. 375–405.

[74] M. I. Miller, A. Trouvé, and L. Younes. "Geodesic shooting for computational anatomy". In: *Journal of mathematical imaging and vision* 24.2 (2006), pp. 209–228.

[75] D. D. Holm and J. E. Marsden. "Momentum maps and measure-valued solutions (peakons, filaments, and sheets) for the EPDiff equation". In: *The breadth of symplectic and Poisson geometry*. Springer, 2005, pp. 203–235.

[76] M. Farias, R. Teixeira, J Koiller, and A. Santos. “Brief review of uncertainty quantification for particle image velocimetry”. In: *Journal of Physics: Conference Series*. Vol. 733. 1. IOP Publishing. 2016, p. 012045.

[77] E. Zhang, K. Mischaikow, and G. Turk. “Vector field design on surfaces”. In: *ACM Transactions on Graphics (ToG)* 25.4 (2006), pp. 1294–1326.

[78] M. Fisher, P. Schröder, M. Desbrun, and H. Hoppe. “Design of tangent vector fields”. In: *ACM transactions on graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 56.

[79] J. Palacios and E. Zhang. “Rotational symmetry field design on surfaces”. In: *ACM Transactions on Graphics (TOG)* 26.3 (2007), p. 55.

[80] F. Knöppel, K. Crane, U. Pinkall, and P. Schröder. “Globally optimal direction fields”. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 59.

[81] K. Crane, M. Desbrun, and P. Schröder. “Trivial connections on discrete surfaces”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1525–1533.

[82] R. Abraham, J. E. Marsden, and T. Ratiu. *Manifolds, tensor analysis, and applications*. Vol. 75. Springer Science & Business Media, 2012.

[83] C. Foiasa, D. D. Holmb, and E. S. Titic. “The Navier–Stokes-alpha model of fluid turbulence”. In: *Physica D* 152.153 (2001), pp. 505–519.

[84] D. W. Thompson et al. “On growth and form.” In: *On growth and form.* (1942).

[85] M. I. Miller, A. Trouvé, and L. Younes. “Hamiltonian systems and optimal control in computational anatomy: 100 years since D’Arcy Thompson”. In: *Annual review of biomedical engineering* 17 (2015), pp. 447–509.

[86] S. Kobayashi and K. Nomizu. *Foundations of differential geometry. Vol. I.* Wiley Classics Library. Reprint of the 1963 original, A Wiley-Interscience Publication. John Wiley & Sons, Inc., New York, 1996, pp. xii+329. ISBN: 0-471-15733-3.

[87] D. D. Holm, J. E. Marsden, and T. S. Ratiu. “Euler-Poincaré models of ideal fluids with nonlinear dispersion”. In: *Phys. Rev. Lett.* 349 (1998), pp. 4173–4177.

[88] S. Chen, C. Foias, D. D. Holm, E Olson, E. S. Titi, and S. Wynne. “The Camassa–Holm equations and turbulence”. In: *Physica D: Nonlinear Phenomena* 133.1 (1999), pp. 49–65.

[89] I. Kolár, J. Slovák, and P. W. Michor. “Natural operations in differential geometry”. In: (1999).

[90] C. J. Cotter and D. D. Holm. “Continuous and discrete Clebsch variational principles”. In: *Foundations of Computational Mathematics* 9.2 (2009), pp. 221–242.

[91] T. Pfaff and N. Thuerey. “mantaflow”. In: *http://mantaflow.ethz.ch/index.html* (2009). URL: `http://mantaflow.ethz.ch/index.html`.

[92] F. de Goes, C. Wallez, J. Huang, D. Pavlov, and M. Desbrun. “Power particles: an incompressible fluid solver based on power diagrams”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 50.

[93] B. Gris, S. Durrleman, and A. Trouvé. “A sub-Riemannian modular approach for diffeomorphic deformations”. In: *International Conference on Networked Geometric Science of Information*. Springer. 2015, pp. 39–47.

[94] J. P. Lewis, M. Cordner, and N. Fong. “Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 165–172.

[95] D. D. Holm, M. Nitsche, and V. Putkaradze. “Euler-alpha and vortex blob regularization of vortex filament and vortex sheet motion”. In: *Journal of Fluid Mechanics* 555 (2006), pp. 149–176.

[96] R. Bridson. “Fast Poisson disk sampling in arbitrary dimensions.” In: *SIGGRAPH sketches*. 2007, p. 22.

[97] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. ISBN: 9780132125895.

[98] R. L. Bishop. “There is more than one way to frame a curve”. In: *The American Mathematical Monthly* 82.3 (1975), pp. 246–251.

[99] P. Farrell, K. Gillow, and H. Wendland. “Multilevel interpolation of divergence-free vector fields”. In: *IMA Journal of Numerical Analysis* 37.1 (2017), pp. 332–353.

[100] H. Bhattacharya, M. B. Nielsen, and R. Bridson. “Steady State Stokes Flow Interpolation for Fluid Control.” In: *Eurographics (Short Papers)*. 2012, pp. 57–60.

[101] J. F. Hughes, A. Van Dam, J. D. Foley, and S. K. Feiner. *Computer graphics: principles and practice*. Pearson Education, 2014.

[102] G. Lorentz. *Bernstein Polynomials*. 1953.

[103] K. Spitzmüller. “Partial derivatives of Bezier surfaces”. In: *Computer-Aided Design* 28.1 (1996), pp. 67–72.

[104] K. Museth, J. Lait, J. Johanson, J. Budsberg, R. Henderson, M. Alden, P. Cucka, D. Hill, and A. Pearce. “OpenVDB: an open-source data structure and toolkit for high-resolution volumes”. In: *Acm siggraph 2013 courses*. ACM. 2013, p. 19.

[105] J. Solomon, M. Ben-Chen, A. Butscher, and L. Guibas. “As-Killing-As-Possible Vector Fields for Planar Deformation”. In: *Computer Graphics Forum*. Vol. 30. 5. Wiley Online Library. 2011, pp. 1543–1552.

[106] F. Losasso, F. Gibou, and R. Fedkiw. “Simulating Water and Smoke with an Octree Data Structure”. In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH ’04. Los Angeles, California: ACM, 2004, pp. 457–462. DOI: 10.1145/

1186562.1015745. URL: http://doi.acm.org/10.1145/1186562.1015745.

[107] C. J. Cotter, J. Frank, and S. Reich. "The remapped particle-mesh semi-Lagrangian advection scheme". In: *Quarterly Journal of the Royal Meteorological Society* 133.622 (2007), pp. 251–260.

[108] A. Angelidis. "Multi-scale vorticle fluids". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 104.

[109] D. D. Holm and C. Tronci. "Multiscale turbulence models based on convected fluid microstructure". In: *Journal of Mathematical Physics* 53.11 (2012), p. 115614.

[110] K. Åhlander. "Supporting tensor symmetries in EinSum". In: *Computers & Mathematics with Applications* 45.4 (2003), pp. 789–803.

[111] A. Cardona, M. Storti, and C. Zuppa. "A tensor library for scientific computing". In: *Journal Mecánica Computacional (AMCA)* (2008).

[112] J. Reinders. *Intel Threading Building Blocks*. First. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007. ISBN: 9780596514808.

[113] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.