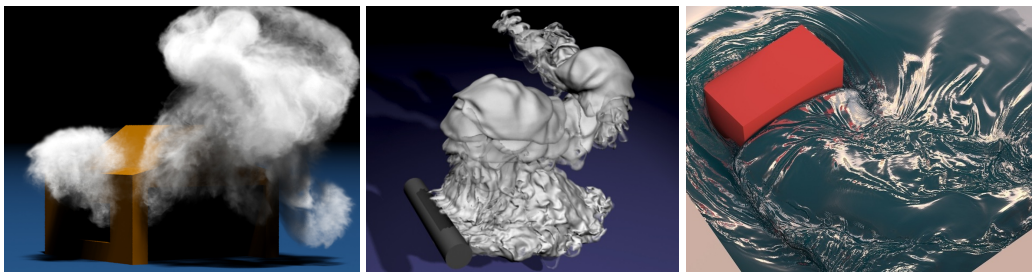


Turbulent Fluids

Tobias Pfaff, UC Berkeley
Theodore Kim, UC Santa Barbara
Nils Thuerey

Contents

1 Introduction **5**
- 1.1 About the Authors . . . 6
- 1.2 Structure of the Course Notes . . . 7

2 Turbulent Flows **8**
- 2.1 The Reynolds Average . . . 8
- 2.2 Turbulence Modeling . . . 10
  - 2.2.1 Energy Transport Models . . . 10
  - 2.2.2 Extending Energy Transport Models . . . 11
- 2.3 The Energy spectrum . . . 13
- 2.4 Turbulence synthesis . . . 15
  - 2.4.1 Curl Noise Synthesis . . . 16
  - 2.4.2 Composition . . . 17
- 2.5 Discussion . . . 19

3 Literature **21**
- 3.1 History . . . 21
- 3.2 Fluid Simulation in Computer Graphics . . . 22
  - 3.2.1 Low-dissipative Methods . . . 22
  - 3.2.2 Sub-grid Methods . . . 22
- 3.3 Lagrangian Vortex Methods . . . 23
- 3.4 Turbulence Methods . . . 25
  - 3.4.1 Turbulence Modeling . . . 25
  - 3.4.2 Turbulence Synthesis . . . 26
- 3.5 Recent works . . . 26

4 Adding Turbulent Detail **28**
- 4.1 Wavelet Turbulence . . . 28
  - 4.1.1 Procedural Wavelet Turbulence . . . 29
  - 4.1.2 High-Resolution Fluid Synthesis . . . 31
  - 4.1.3 Results . . . 34
  - 4.1.4 Conclusions . . . 35
- 4.2 Anisotropic Turbulence Modeling . . . 37
  - 4.2.1 Overview . . . 38
  - 4.2.2 Turbulence Model . . . 39

4.2.3 Implementation . . . 44
4.2.4 Results and Discussion . . . 47
4.2.5 Conclusions . . . 50
4.3 Obstacle-Induced Turbulence . . . 52
4.3.1 Overview . . . 53
4.3.2 Vorticity Formulation . . . 54
4.3.3 Wall-Induced Turbulence . . . 56
4.3.4 Turbulence Synthesis . . . 60
4.3.5 Implementation . . . 64
4.3.6 Results and Discussion . . . 67
4.3.7 Conclusions . . . 69
4.4 Buoyant Turbulence . . . 73
4.4.1 Vortex primitives . . . 74
4.4.2 Vortex Sheet Methods . . . 78
4.4.3 Wall-based Turbulence Model . . . 81
4.4.4 Implementation . . . 84
4.4.5 Results . . . 87
4.4.6 Conclusion . . . 89
4.5 Conclusions . . . 91
4.6 Application Guidelines . . . 91

5 Liquid Turbulence **93**
5.1 Wave Models . . . 94
5.1.1 The Classic Wave Equation . . . 94
5.1.2 The Korteweg-de Vries Equation . . . 94
5.1.3 The Non-Linear Schrödinger Equation . . . 94
5.2 The iWave Algorithm . . . 95
5.2.1 The iWave Equation . . . 95
5.2.2 Spatial Discretization . . . 96
5.2.3 Time Discretization . . . 98
5.2.4 A Preview of the 3D iWave Kernel . . . 99
5.3 Closest Point Turbulence . . . 99
5.4 Previous Work . . . 101
5.5 A Free Surface Turbulence Algorithm . . . 102
5.5.1 The Closest Point Method . . . 102
5.5.2 The iWave Algorithm . . . 103
5.5.3 Building a 3D Vertical Derivative . . . 104
5.5.4 Reducing Projection Error . . . 106
5.5.5 A Fast Closest Point Transform . . . 107
5.5.6 Building and Advecting the Extension Field . . . 108
5.5.7 Turbulence Seeding . . . 112
5.5.8 The Complete Algorithm . . . 113
5.6 Discussion and Results . . . 113
5.7 Conclusions and Future Work . . . 117

Chapter 1

Introduction

Over the last decade, the special effects industry has embraced physics simulations as a highly useful addition to its tool-set for creating realistic scenes ranging from a small camp fire to the large scale destruction of whole cities. The simulation methods used to create these effects are largely based on techniques originally developed to replace scientific experiments with computer simulations. In a direct application of this paradigm to movie making, we can now replace a real effects set, such as the staging an exploding house, with the simulated explosion of a virtual model of the house. This has some obvious advantages: it is more cost-effective, enables a wider variety of effects and of course it is far less dangerous for the people involved. However, the desire to fine-tune and control effects in general is arguably the primary reason why movie makers prefer the use of virtual tools over their traditional counterparts. Unfortunately, controlling the details of a violent phenomenon such as an explosion remains problematic even using numerical simulations. Due to the chaotic nature of turbulent fluids, such simulations tend be both computationally expensive and unpredictable. Small changes in initial conditions or a change of resolution will produce unexpected changes in the final motion, and make it hard for animators to obtain the desired behavior for the effect. For this reason, the following course notes will focus on tools for augmenting existing coarse simulations with turbulent detail. This enables rich detail and visually interesting small-scale motion, but also allows for a practical multi stage work flow that gives artists control over large scale motion and small scale details separately.

Overall, this course aims at providing an overview and practical guidelines to employing turbulence modeling techniques for fluid simulations. Turbulence has been a topic of research in classical fluid dynamics for a long time, and is discussed in a vast body of publications. This course will give a condensed overview of the central concepts, and introduce modeling techniques that are relevant for applications in Computer Graphics. More specifically, control and art direction of simulations are enabled with a two stage work flow - first, a rough initial simulation is conducted. In a second stage, turbulent effects are computed and applied to the simulation to increase its detail level. Motivated by these concepts, several approaches for increasing the visual detail of fluid simulations will be introduced. In addition to discussing single phase simulations, e.g. smoke and fire, we will also discuss the difficulties surrounding multi-phase liquid turbulence, and present a practical new algorithm for its simulation.

As a central aim of this course is to provide information on how to use turbulence theory

for practical applications, source code examples for the methods covered will be made available. Additionally, the implementations will be discussed to provide starting points for navigating the source code.

The goal is to give developers interested in implementing powerful fluid solvers the knowledge to apply turbulence models, and to give artists who are curious about the technology a better understanding of when and how to make use of the different methods. This naturally also includes knowledge of the limitations of the various approaches; this course therefore also provides guidelines and a discussion of the important pros and cons for each of the introduced methods. While the course notes are structured based on papers in the field, most of the presented methods are modular. We encourage mixing and matching predictor and synthesis components from the various methods to find the best solution to a given problem.

1.1 About the Authors

The three lecturers of this course have worked in the field of physically-based animation and especially turbulence methods for fluid simulations over the course of many years. They have made central contributions to the field, several of which are now used in professional tools and feature film productions.

- **Theodore Kim** has been an Assistant Professor in the Media Arts and Technology Program at the University of California, Santa Barbara since 2011. He received his PhD in Computer Science from the University of North Carolina, Chapel Hill in 2006, and subsequently held Post-Doctoral positions at IBM TJ Watson Research Center and Cornell University. He was an Assistant Professor in Computer Science at the University of Saskatchewan from 2009 to 2011. His research investigates the aesthetic and engineering possibilities of simulated physics, such as fluid dynamics, virtual humans, snowflake growth, and lightning formation. He received a Scientific and Technical Academy Award in 2012, and an NSF CAREER Award in 2013.
- **Tobias Pfaff** is a Post-Doctoral researcher in the Computer Graphics Lab at the University of California, Berkeley. In 2012, he received his PhD from ETH Zurich for his dissertation on turbulence methods in Computer Graphics. He also has a background in Physics, and obtained a MSc in Physics from the University of Constance in 2007. His main research interest is physical simulation, with a focus on turbulent fluid systems and thin shell dynamics. He is the author of the open-source fluid simulation framework Mantaflow, which implements his published methods.
- **Nils Thuerey** currently has a position as research & development lead at ScanlineVFX, working on the design and implementation of large-scale physics simulators for visual effects. His research focuses on physically-based animation, with a particular emphasis on detailed fluids and turbulence. He received his Ph.D (with honors) in 2007 from the University of Erlangen-Nuremberg, and worked as a post-doctoral researcher with Ageia/Nvidia and the Computer Graphics Laboratory of ETH Zurich until 2010. Several of his algorithms are now available in software packages such as Houdini or Blender, and his work can be seen in movies such as *Avengers*, *Iron Man 3* and *Man of Steel*.

1.2 Structure of the Course Notes

This document is divided into three main parts. The first part focuses on a comprehensive overview of the theory of turbulence and turbulence modeling from the field of computational fluid dynamics. In the second part we will review commonly used solvers for fluid simulations in graphics, and discuss the popular wavelet turbulence approach for increasing the resolution of simulations. Equipped with these tools, more powerful turbulence models will be introduced, which enable modeling of nontrivial turbulent flows, e.g. turbulence with strong directional preferences, or turbulence triggered by temperature differences in the flow. In the third and last part, we will discuss the difficulties that are encountered when dealing with liquids, and we will describe how to apply the ideas of scale separation and controllable detail to liquid simulations.

Chapter 2

Turbulent Flows

Many complex flows, ranging from chimney smoke and explosions to the wake of a ship in the ocean, show chaotic and irregular vortical flow disturbances. These flows are called *turbulent flows*. Compared to *laminar* flows, such as slow-moving rivers, or the air flow field over a candle, turbulent flows show an abundance of detail on many length scales. While this detail constitutes the appealing look of many flow phenomena, representing it directly in the simulation requires enormous storage and computing capabilities. It is however possible to describe turbulence in terms of their statistical properties. Turbulence modeling theory aims exactly at that. In this section, established turbulence theory is introduced which forms the basis for the methods described in this course. A detailed overview of turbulence modeling theory can also be found in the books by Pope [Pop00] and Wilcox [Wil93].

2.1 The Reynolds Average

To provide a measure of the overall strength of turbulence in the flow, the *Reynolds number*

$$\mathrm{Re} = \frac{vL}{\nu} \tag{2.1}$$

is used. Flows with a Reynold number below 1500 are typically laminar, while a Reynolds numbers over 5000 are a strong indicator for a turbulent flow [Pop00]. The quantities used here are the flow velocity v, the fluid viscosity ν and the characteristic lengthscale L. The definition of L depends on the problem at hand, for pipe flow it would e.g. correspond to the pipe diameter.

While the Reynolds number provides a general estimate on turbulence behavior, most flows are not completely turbulent or laminar. A river with an immersed obstacle, for example, may be laminar in most regions, but show transition to turbulence in the wake of the obstacle. To study the behavior of turbulence, it is therefore beneficial to decompose the flow field in a turbulent and mean flow component. The *Reynolds decomposition* achieves this by introducing a mean velocity field by the average $\bar{\mathbf{U}} = \langle \mathbf{u} \rangle$. The remaining component $\mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle$ then describes the turbulent fluctuation. It can be shown that both components remain divergence-free [Pop00].

The averaging operator $\langle \cdot \rangle$ introduced above is interpreted in the sense of an expectation value of a random field. Its concrete realization, and therefore the concrete classification of

turbulence and mean component depends on the type of problem investigated. For most applications, an average over ensembles, time or lengthscale is used. The equivalence of these interpretations for large sample numbers is given by the ergodicity theorem.

RANS The motion equations for Newtonian fluids in its continuum formulation are called the Navier-Stokes (NS) equations. They are a set of partial differential equations in the velocity field $\mathbf{u}(\mathbf{x},t)$. For the case of incompressible fluids, they are written as

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

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \frac{1}{\rho}\mathbf{g} \quad , \tag{2.3}$$

with the fluid viscosity ν, density ρ, and the gravity $\mathbf{g}$. The pressure field is denoted by p.

Using the mean flow definition, it is possible to derive properties for the mean and the fluctuating component separately. If we apply the ensemble average operator to the Navier-Stokes equation, we obtain the motion equation for the mean component, the *Reynolds-averaged Navier-Stokes equation (RANS)*

$$\frac{\mathrm{D}\bar{\mathbf{U}}}{\mathrm{d}t} = \nu\nabla^2\bar{\mathbf{U}} - \rho(\nabla\cdot\boldsymbol{\tau}) - \frac{1}{\rho}\nabla\langle p\rangle \quad . \tag{2.4}$$

This equation is identical to the Navier-Stokes momentum equation, except for the additional stress term $\rho(\nabla\cdot\boldsymbol{\tau})$. The symmetric tensor $\tau_{ij} = \langle u_i' u_j'\rangle$ is called the *Reynolds stress tensor*, and describes the influence of turbulent fluctuations on the mean flow field.

The RANS equation is used in many engineering applications, as it allows to predict the impact of small-scale turbulent detail on e.g. the mean flux in an pipe or engine, without directly simulating it. Unfortunately, the Reynolds stress tensor still depends on the fluctuating components, and cannot be expressed in terms of averaged properties. This *closure problem* in CFD is solved by additional assumptions and models of the behavior of turbulence, and has given rise to an area of research, namely *turbulence modeling*.

Turbulent Viscosity Hypothesis To investigate the effect of the Reynolds stress tensor in (2.4), it is helpful to split the Reynolds tensor into a isotropic and and anisotropic part. The isotropic component can be expressed in terms of the *turbulent kinetic energy*, that is the energy contained in turbulent fluctuations. It is defined as

$$k = \frac{1}{2}\langle \mathbf{u}'\cdot\mathbf{u}'\rangle \quad . \tag{2.5}$$

The isotropic component now becomes a diagonal tensor $\frac{2}{3}k\delta_{ij}$ and can therefore be expressed as a scalar. This means its effect in (2.4) is that of a pressure, and it can easily be absorbed in an effective pressure $\langle p\rangle_E = \langle p\rangle + \frac{2}{3}k$. The anisotropic component, on the other hand, is defined as $a_{ij} = \tau_{ij} - \frac{2}{3}k\delta_{ij}$ and still needs to be modeled. The *turbulent viscosity hypothesis* assumes that its effects is purely viscous. This is a reasonable approximation, as the superposed small-scale movements act as a diffusion on larger scales. The turbulent viscosity hypothesis is therefore expressed in analogy to viscous stress by

$$a_{ij} = 2\,\rho\,\nu_T S_{ij} \tag{2.6}$$

with the scalar turbulent viscosity ν_T and the mean strain-rate tensor $S_{ij} = \frac{1}{2}(\frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i})$. If we substitue the Reynolds tensor and the effective pressure and turbulent viscosity in (2.4), we obtain

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla\langle p\rangle_E + \nabla((\nu + \nu_T)\nabla\mathbf{u}) \tag{2.7}$$

which has the form of the NS equation (2.3) with modified pressure and the increased viscosity term $\nu + \nu_T$. We have to note, though, that ν_T is in general a function of space and time, while the molecular viscosity ν is a constant.

If the turbulent viscosity hypothesis is used, the problem of modeling the Reynolds stress tensor reduces to modeling the scalar turbulent viscosity ν_T. Most classic turbulence models are based on this assumption. However, it has to be noted that turbulent viscosity is only an approximation, and cannot describe certain effects, such as anisotropic turbulence generation. Turbulence models based on turbulent viscosity are discussed in the next chapter. A short outlook on including anisotropic effects will also be provided.

2.2 Turbulence Modeling

Turbulence modeling tries to predict properties of the fluctuating turbulence based on the mean velocity field. For averaged simulations such as RANS, the important variable to model is the Reynolds stress tensor, and therefore a big part of existing research in turbulence focuses on predicting the Reynolds stress as accurate and general as possible. In this section, classical turbulence modeling based on turbulent viscosity is introduced. This is what is used in most CFD simulation packages for engineering, and it also forms a basis for many of the methods presented in this course. There are also other, fundamentally different approaches for describing turbulence, most notably stochastic pdf methods [Pop83] and Large Eddy Simulations (LES) which are used heavily in e.g. meteorology. An overview of LES methods can be found in [GO93] and [Joh06].

2.2.1 Energy Transport Models

There are different approaches for modeling turbulent viscosity, most of which are based on empirical assumptions. These methods can be classified by the number of model variables and in terms of their completeness, that is whether they require scene-dependent constants or fields. The simplest conceivable model is assuming ν_T to be constant across the flow. This model is limited to very simple flows, and does not provide much insight over directly specifying turbulence energy. It is thus not useful for any practical application.

A better approach is to model ν_T in terms of a mixing length. These models can be accurate if the mixing length of the respective problem is known, and have successfully been used in aerospace engineering [BL78]. Other models use the fact that turbulence properties are well described by advection-diffusion processes, and model these processes using one or more partial differential equations. Due to their generality, these models are among the most common turbulence models. Below, a mixing-length model and one common two-equation model are presented.

Mixing length model The *mixing length model* is based on a generalization of the explicit turbulent viscosity term from boundary layer flows. Baldwin et al. [BL78] suggest the definition

$$\nu_T = l_m^2 \|\mathbf{\Omega}\| \tag{2.8}$$

where l_m is the characteristic mixing lengthscale, and $\Omega_{ij} = \frac{1}{2}(\frac{\partial \bar{U}_i}{\partial x_j} - \frac{\partial \bar{U}_j}{\partial x_i})$ the rotation tensor of the mean flow field. The mixing length encodes the geometry of the problem, and the accuracy of the model largely depends on the correct specification of this length. Analytic expressions for l_m are known for a certain type of problems, such as its linearity in wall distance in the log-law region of boundary layers. On the other hand, for the general case far from boundary layers, the mixing length behavior is largely unknown. Therefore, this model is considered *incomplete*. This is model used in [PTSG09, NSCL08].

k–ε model The k–ε model uses a similar definition of turbulent viscosity as mixing length models, but expresses it in terms of turbulent kinetic energy k and turbulence dissipation ε as

$$\nu_T = C_\mu \frac{k^2}{\varepsilon} \quad . \tag{2.9}$$

The modeling constant C_μ is defined as 0.09 from empirical observation [LS74]. An evolution equation for the variables k, ε could theoretically be obtained by a Reynolds-average of the Navier-Stokes equation in the same way as (2.4). As this equation contains mainly terms that cannot be derived from mean flow properties, however, the implication of the individual terms is studied and modeled. The complete PDE system in this model is defined as

$$\begin{aligned} \frac{\mathrm{D}k}{\mathrm{d}t} &= \nabla(\frac{\nu_T}{\sigma_k}\nabla k) + \mathscr{P} - \varepsilon \\ \frac{\mathrm{D}\varepsilon}{\mathrm{d}t} &= \nabla(\frac{\nu_T}{\sigma_\varepsilon}\nabla \varepsilon) + \frac{\varepsilon}{k}(C_1 \mathscr{P} - C_2 \varepsilon) \quad . \end{aligned} \tag{2.10}$$

The terms $\mathscr{P}$ and ε denote production and dissipation of turbulent kinetic energy respectively, while the modeling constants are specified as $\sigma_k = 1$, $\sigma_\varepsilon = 1.3$, $C_1 = 1.44$ and $C_2 = 1.92$. It can be observed that both equations consist of a turbulent diffusion term in analogy to the RANS equation (2.4) as well as advection and production and dissipation terms. The implicit advection contained in the substantial derivative on the left-hand side refers to advection in the mean flow field $\tilde{\mathbf{U}}$. The production term depends on the strain rate of the mean flow field and is deduced as

$$\mathscr{P} = 2\,\nu_T \|\mathbf{S}\|^2 \quad . \tag{2.11}$$

The equation system is now fully specified and only depends on properties of the averaged flow field, and is therefore considered *complete*. It is due to this generality that the k–ε model is among the most commonly used turbulence models in CFD.

2.2.2 Extending Energy Transport Models

Turbulence models are, due to their semi-empirical nature, only accurate under certain conditions. A multitude of turbulence models exist, and is used depending on the scenario.

For example, the k–ε model performs well for shear flows with small pressure gradient – for strong pressure gradients, the k-ω model is superior, but has other drawbacks. In general, RANS turbulence modeling is considered much less accurate than using Reynolds stress transport models, pdf methods or LES. On the other hand, it is by far the best understood approach, easy to implement and most importantly, computationally inexpensive. For the detail level desired in typical Graphics applications, LES for example does not gain much over direct simulation in terms of performance. Therefore, RANS is still the most commonly employed method, even in CFD where accuracy is of more importance than in Graphics.

Stability To address some of the shortcomings of RANS-based turbulence methods, model extensions have been proposed. For our purposes, the most vital issues is to address to instability of the k–ε model for low values of k and ε. This model therefore requires a minimal turbulence intensity to be present. But even a simulation with high turbulence level may become instable in wall regions, as the viscous sublayer very close to the wall drives the effective Reynolds number to zero. The k–ε model is therefore often extended by *Low-Reynolds models*, which consist of additional dampening terms that are active in near-wall regions. For e.g. real-time simulations with large timesteps, however, even this may however not be sufficient as the simulation can still easily become instable. An alternative is a clamping system that restricts the parameters to a meaningful range. Such a system is described in Section 4.2.

Reynolds Stress Transport Models A further limitation of all RANS-based turbulence models is their reliance of the turbulent viscosity hypothesis. The assumption implies that the Reynolds tensor is aligned to the mean flow strain-rate field, which is not the case for e.g. flows with fast varying mean flow. It also provides few information on the turbulence anisotropy, which would prove useful for turbulence synthesis. Reynolds stress transport models avoid this limitation by solving a partial differential equation system for the complete Reynolds stress tensor, instead of the energy k. The model is written as

$$\frac{\mathrm{D}\langle u_i u_j \rangle}{\mathrm{d}t} + \sum_k \frac{\partial}{\partial s_k}(T^{\nu}_{kij} + T^{p}_{kij} + T^{u}_{kij}) = \mathscr{P}_{ij} + \mathscr{R}_{ij} - \varepsilon_{ij} \quad . \tag{2.12}$$

The transfer tensors T^{ν}_{kij},T^{p}_{kij},T^{u}_{kij} describe viscous diffusion, pressure transport and turbulent convection, respectively. $\mathscr{P}_{ij}$ and ε_{ij} are the production and dissipation tensors, in analogy to the scalar terms introduced in Section 2.2.1. The most interesting difference compared to energy transfer models is the appearance of the *redistribution term* $\mathscr{R}_{ij}$. Redistribution characterizes the transfer of energy between between isotropic and anisotropic components of turbulence. The major effect in this process is *isotropization*: Turbulence generated from the mean flow is usually highly anisotropic, and over time driven towards isotropy by energy exchange. The most commonly used realization of $\mathscr{R}_{ij}$ is the LRR-IP model [LRR75]

$$\mathscr{R}_{ij} = -C_R \frac{\varepsilon}{k}(\langle u_i u_j \rangle - \frac{2}{3} k \delta_{ij}) - C_I(\mathscr{P}_{ij} - \frac{2}{3}\mathscr{P}\delta_{ij}) \tag{2.13}$$

with the model constants $C_R = 1.8$ and $C_I = 0.6$. Pfaff et al. [PTC+10] make use part of this model to augment an energy transfer model for anisotropy awareness. This will be explained in Section 4.2.

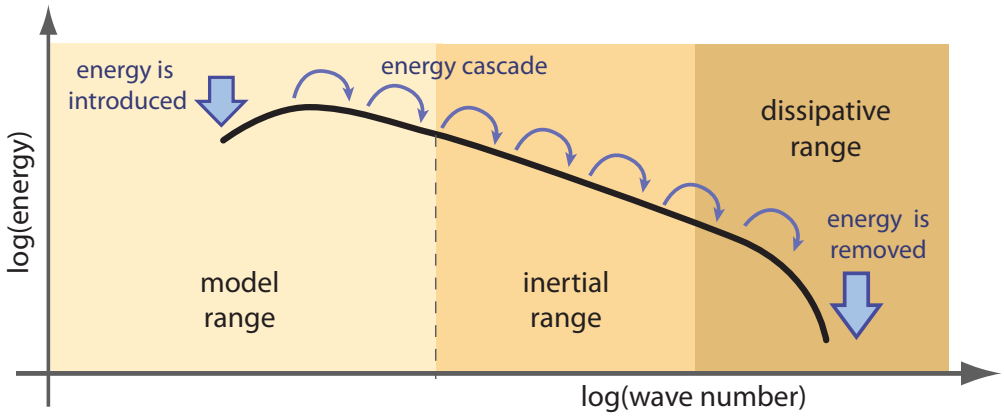


Figure 2.1: This graph shows the typical evolution of the energy per vortex wavenumber. Energy is introduced into the system at large scales in the model range. The energy is subsequently transferred into smaller scales by scattering of vortices, and finally dissipates due to viscosity in the dissipative range.

2.3 The Energy spectrum

In the previous sections the spatial distribution of turbulent kinetic energy was described. We will now investigate turbulence in terms of its spectral distribution. One way to approach this is the solution of the NS equation in the frequency domain, e.g. [dFN01]. These spectral methods have some desirable properties, such as fast convergence, and drawbacks such as difficulties in boundary geometry handling. In terms of turbulence, however, they do not provide more insight than their time domain counterparts. Instead, in this chapter the spectral distribution of turbulent kinetic energy as defined in Section 2.1 is studied.

Turbulent length scales We can think of turbulent length-scales as the size of the eddies that compose the turbulence. From experiments we can observe that while turbulent fluctuations occur on many length scales, its behavior is very different on these scales. For high-Reynolds number flows, it is observed that turbulent energy is generated mainly on large scales and dissipated on small scales. The Richardson interpretation of this states that large eddies are instable, and break up to form smaller eddies until they are eventually dissipated to heat by viscous processes[1]. This results in an *energy cascade*, that is the transfer of turbulent kinetic energy from large to small scales. Following this notion, the energy spectrum can be divided into three regimes. This is also illustrated in Fig. 2.1.

- *Model range*. In this region, large-scale structures are dominant and most of the spectrum's energy is contained. Its behavior is strongly dependent on the flow geometry, and is therefore not easily described by statistical models. The production of turbulence mainly occurs in the model-dependent range by strain processes acting on the mean flow.

[1] *"Big whorls have little whorls, which feed on their velocity, and little whorls have lesser whorls, and so on to viscosity."* – Lewis Richardson

- *Inertial subrange*. This range shows very little production and dissipation. The main active process being forward scattering, that is the transfer of energy from small to high wave numbers.
- *Dissipation range*. The main energy dissipation occurs in the range of large wave numbers. This is driven by molecular viscosity, which is active for very small structures typically below the millimeter range.

Kolmogorov's law In his famous work, Kolmogorov [Kol41] proposed that for high-Reynolds number flows, fully-developed turbulence can be described very easily in a statistical sense. While large eddies are in general anisotropic and strongly influenced by boundary conditions, this behavior is lost by the energy exchange in forward scattering. Turbulence in the inertial subrange and the viscous range can thus be assumed to be *locally isotropic* and the statistical turbulent behavior in this regime is fully determined by the dissipation ε and viscosity ν. Kolmogorovs hypothesis further states that energy spectrum in the inertial subrange can be described as

$$E(\kappa) = C\varepsilon^{\frac{2}{3}}\kappa^{-\frac{5}{3}} \tag{2.14}$$

with the wavenumber κ and constant C. This is referred to as *Kolmogorovs five-thirds law* or *K41 theory*.

Beyond Kolmogorov While Kolmogorovs law is a useful tool to describe fully-developed turbulence, transition to turbulence and larger scale turbulence are not covered by K41 theory. There are, however, extensions to this model. Most notably, the energy spectrum can be extended to cover the dissipation range as well without losing too much of its generality [Pao65]. This regime is not very interesting for Graphics though, as it is situated on a length scale well below the desired resolution for most Graphics simulations. The model-dependent range, on the other hand, is very hard to describe statistically in a general way. Even if such an averaged energy spectrum existed, its expressiveness would be limited, as the dynamics in the model range are dominated by anisotropic, coherent structures and their interplay with boundary conditions. An accurate description of this regime will therefore necessarily have to track individual structures.

Another possibility of obtaining more generality is to model the energy evolution of the energy spectrum, instead of considering an stationary spectrum. This can describe some of the effects of turbulent transition. *Spectral energy transfer* models approach this in the same manner as the spatial transfer models introduced in Section 2.2.1. In contrast to spatial transfer models, the individual terms are however not as easy to model, and result in complex and instable systems. Therefore, these models are mainly used to derive stationary spectra instead of transient modeling [Pop00]. A review on spectral energy transfer methods can be found in [Pan71].

This being said, it is possible to describe the behavior outside K41 using additional data. One possibility to do this is modeling a subset of anisotropic production [PTC+10].

2.4 Turbulence synthesis

For CFD applications, the interest in turbulence is mainly focused on averaged properties: its influence of turbulence on the mean velocity, turbulent mixing or the induced forces. In Computer Graphics, however, the transient behavior of turbulent detail itself is important, as it makes out the desired visual appearance. Based on the observations in Section 2.3 that fully-developed turbulence has a rather uniform behavior, it is possible to generate detail without a costly full simulation. This *turbulence synthesis* generates detail that obeys certain statistical properties predicted by CFD turbulence models, such as the ones described in the previous sections. Detail generated this way will therefore correspond to detail actually observed in high-resolution reference simulations or real-world experiments only in a statistical sense.

The most common approach is to synthesize a high-resolution velocity field to represent the additional detail. There are however different statistics that can be used for synthesis, and different ways to generate detail with given statistical properties. This does not mean all of these realizations will produce realistic output, though. Since turbulence generation does not imply the fulfillment of the NS equations, there is no guarantee that a given realization of a statistical representation will behave like a fluid. This means that to obtain believable results, additional information beyond the spatial distribution and the frequency spectrum are necessary. The goal of turbulence synthesis is therefore to find a combination of conditions that produces turbulent detail in a believable manner. In this section, we will introduce the commonly used curl noise synthesis. Other synthesis approaches include random forcing [ZYC10] or synthesis using Lagrangian vorticity primitives [PTSG09].

Believable detail To our knowledge, there is no direct comparative study on which properties are exactly required for realistic appeal of turbulence. In our experience, the qualities listed below are vital in order achieve realism.

1. *Solenoidality*. The most distinguished property of fluid flows is their solenoidal behavior (2.2). It is responsible for the swirly look of fluids, especially on the small scales. Thus, it is vital that the generated detail velocity field remains divergence-free in order to produce realistic results.
2. *Temporal coherence*. The temporal continuity is of equal importance as discontinuities between timesteps will cause visible artifacts in the flow field. Particular care has to be taken that coherent turbulent features such as eddies are preserved over time, otherwise turbulence appears as discoherent noise.
3. *Spectral distribution*. To obtain the characteristic look of turbulence, the distribution of vortex sizes is important. Such distribution can be for example obtained from the Kolmogorov law for the inertial subrange.
4. *Spatial distribution*. In most flows, the turbulence intensity is not homogeneous in whole domain, but areas with strong turbulence and areas with negligible turbulence will exist. This is especially true for flows with high turbulence production, such as flows around obstacles and buoyant plumes, in which turbulence intensity varies strongly. Traditionally, information about spatial distribution of turbulence is extracted from the flow field. This is however only viable for resolutions high enough to resolve turbulence generation. This thesis will introduce an approach that uses

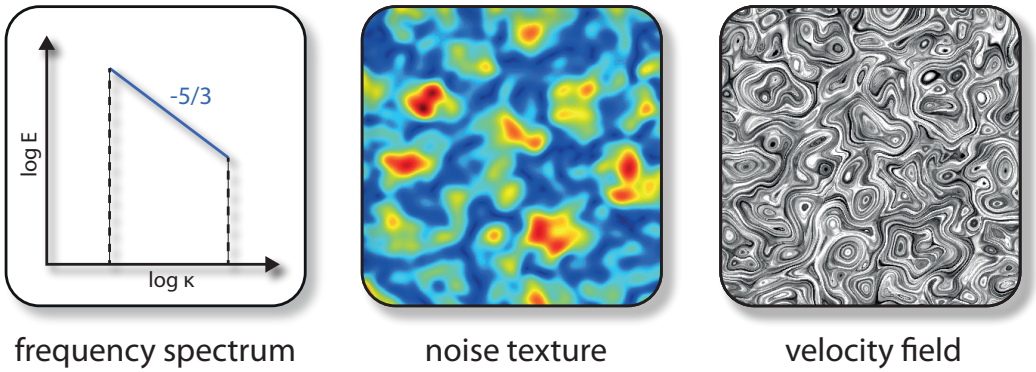


Figure 2.2: Curl noise synthesis for the Kolmogorov spectrum. The energy spectrum is transformed into a noise field using a Fourier transform. Applying the curl operator yields a divergence-free velocity field with the desired frequency behavior.

energy transfer models to predict spatial distribution, therefore allowing lower base resolutions (Section 4.2).

2.4.1 Curl Noise Synthesis

The most straightforward approach to satisfy both solenoidality and a prescribed spectral distribution is curl noise synthesis. In curl noise synthesis, a three-dimensional noise field N_f is synthesized from an energy spectrum. First, the desired spectrum $E(\kappa)$ is overlaid with a random phase $\varphi \in [0 \dots 1]$ and transfered to a spatial noise field using the Fourier transform

$$N_f(x) = \int E(\kappa) \cdot e^{-i\kappa x + i2\pi\varphi(\kappa)} \mathrm{d}\kappa \quad . \tag{2.15}$$

The same effect can be achieved by dividing the energy spectrum into octaves, and stacking multiple octaves of a narrow-band noise field N_i, for example Wavelet Noise [CD05], with the respective energy coefficients E_i

$$N_f(x) = \sum_i E_i N_i(x) \quad . \tag{2.16}$$

This noise field can now be used to generate a detail velocity field. First, three noise fields N_x, N_y, N_z are generated using the same energy spectrum but different phases. These fields can now be interpreted as a vector potential. The detail velocity is then generated by applying the curl operator

$$\mathbf{u}_D(\mathbf{r}) = \nabla \times \sqrt{\alpha_S} \begin{pmatrix} N_x(\mathbf{r}) \\ N_y(\mathbf{r}) \\ N_z(\mathbf{r}) \end{pmatrix} \tag{2.17}$$

with a detail strength coefficient α_S. This process is illustrated in Fig. 2.2 for the Kolmogorov spectrum (2.14). The curl operator guarantees the solenoidality of the resulting velocity field. As the curl operator is linear, the frequency characteristic of the noise field also remains intact. A detailed account on accurately computing this velocity on discrete grids can be found in Kim et al. [KTJG08].

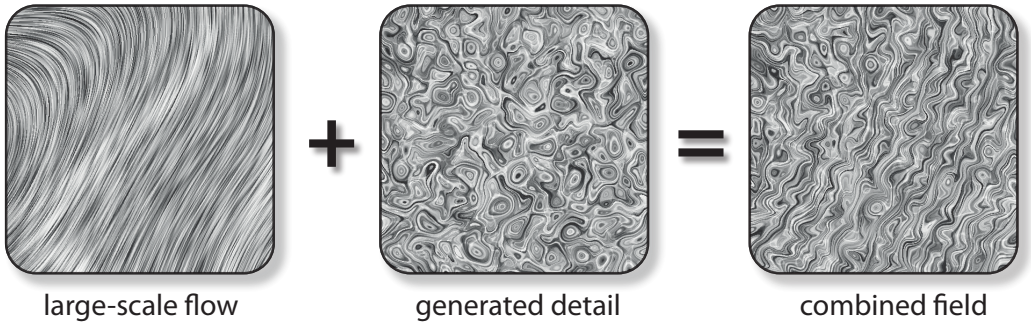


Figure 2.3: A large-scale flow field from a simulation is combined with detail generated by curl noise synthesis.

2.4.2 Composition

After synthesizing a velocity field with the desired frequency spectrum, this field has to be combined with the large-scale simulation to form a coherent flow. First, a separation of the scales for model-dependent large-scale flow and uniform turbulent behavior has to be introduced. In LES, this is realized by applying frequency filters while for RANS simulations, the mean flow field $\bar{\mathbf{U}}$ represents the large-scale simulation. Methods employed in Graphics use a notion similar to RANS. As base simulations with low resolution and a diffusive semi-Lagrangrian advection typically have an inherent diffusion higher than the turbulent diffusion in RANS, the simulation is used directly as the large-scale flow. For synthesis methods based on K41, this assumes the grid resolution marks the division between the model-dependent range and the inertial subrange. The former is then represented by the large-scale flow, while later is obtained by synthesized sub-grid detail.

The most common solution to store sub-grid detail is using a grid of higher resolution than the base simulation. Some methods also operate on the same grid resolution as the base simulation – this makes sense as the frequency cutoff induced by numerical dissipation happens on the scale of the multiple grid cells. However, the improvement in detail obtained this way is obviously limited.

To combine large-scale flow and synthetic detail, these two fields are composed. As a first step, this requires upsampling the large-scale velocity field to the resolution of the detail field. If the frequency spectra of the two fields can be assumed to be disjunct in the sense of RANS, they can be simply added (Fig. 2.3). If, on the other hand, the spectra overlap, care has to be taken that features are not duplicated. This is especially the case if detail field and large-scale flow fields are of the same resolution. In this case, reinforcement techniques are applied: The turbulence intensity is measured on the large-scale field, and compared to the detail field. Only the difference between these fields is then added to form the resulting field. This approach is used e.g. in vorticity reinforcement [SRF05] and vorticity confinement [FSJ01].

Spatial distribution Combining a simulation with a synthesized detail field as above results in uniform, homogeneous turbulence over the complete field. For most scenarios, this is not sufficient, as turbulence intensity will vary over the domain. Here, *turbulence*

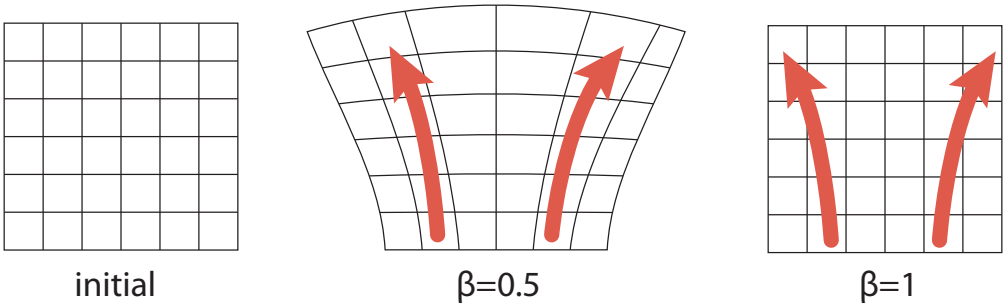


Figure 2.4: The texture coordinate field t_2 is depicted. Initially, the coordinates correspond to position in space. Over time, the field deforms due to mean flow velocity, drawn in red here. The field t_2 is reset on $\beta = 1$, when its coefficient in (2.19) is zero.

predictors are employed to estimate the spatial distribution of turbulence strength. Simple turbulence predictors measure small-scale whirls present in the base simulation by vorticity [FSJ01] or wavelet decomposition [KTJG08] and assume they form the upper level of the turbulent energy cascade. Based on their energy, the lower levels of turbulence can then be reconstructed. This however requires a simulation resolution high enough that turbulence is formed at all. In our methods, we employ energy transfer models as described in Section 2.2.1 to estimate the spatial turbulence intensity.

To incorporate spatially varying fields of turbulence intensity obtained either way into curl noise synthesis, the detail strength coefficient in (2.17) can be modified. Based on (2.14), this can be achieved either via the turbulent kinetic energy or dissipation

$$\alpha_S(\mathbf{r}) \propto k \propto \varepsilon^{\frac{2}{3}} \quad . \tag{2.18}$$

Strictly speaking, this violates solenoidality, as the modulation may introduce divergence. In practice, this is not a problem as long as the gradient of α_S is not too steep, as these divergences do not accumulate over time. For steep gradients, such as interfaces of buoyant plumes, this will however lead to visual artifacts. A more correct synthesis could be achieved by incorporating the spatial intensity distribution directly into the synthesis, for example using wavelets. To our knowledge, this approach has not been used in any synthesis method so far, which is largely due to the numerical complexity of performing a full spectral transfer in each simulation frame.

Temporal coherence One of the most intricate issues in turbulence synthesis is ensuring temporal coherence. This is due to the fact that the two main goals in temporal coherence are incompatible. First, the generated velocity field should deform in accordance with the flow. As the turbulent energy cascade mainly involves forward scattering, it is assumed that the detail field remains passive, and moves within the large-scale flow. It should however also not strongly deform, as this will distort the frequency behavior, and destroy the characteristic turbulent shapes. By advecting the detail field in the large-scale flow, deformations accumulate and will inevitably induce strong deformation. This is also a common problem in texture synthesis, and the solutions are similar in both fields.

The advection in the large-scale flow can be realized using texture coordinates which index positions in the detail field. At the start of the simulation, the texture coordinates will correspond to their position in space. Over the course of the simulation, they are advected in the velocity field which leads to distortion of the field. The most commonly used technique in preventing strong distortions is coordinate resetting. After a number of steps, all coordinates are reset to their position in space. As this will naturally induce jumps in the velocity field, two sets of texture coordinates are used, and reset alternately. The generated velocity at a point is then the linear combination based on its two texture coordinates t_1, t_2.

$$\mathbf{u}' = \beta \mathbf{u}_D(t_1) + (1-\beta)\mathbf{u}_D(t_2) \tag{2.19}$$

with $\beta \in [0,1]$ being a sawtooth function in time. The texture coordinate sets can be reset when its respective coefficient is zero. This process is illustrated in Fig. 2.4. There are also alternative approaches, such as local resetting based on deformation strength as will be presented in Section 4.1.4.

2.5 Discussion

When applying turbulence methods, it is important to realize the limits of the used model and statistic methods in general. This is a point often neglected in turbulence methods for Graphics. Some violations of the limits will not be visible and may be tolerable, due to the fact that the human perception system is not trained to spot inaccuracies in fluid dynamics. Others might severely affect the perceived realism of the scene. This will also heavily depend on scene setup and rendering – for example, the anisotropic turbulent transition region is directly visible for dense smoke, while its effect is less visible for diffuse smoke. Therefore, statistical, isotropic turbulence models as the ones described in the previous chapters are likely to produce artifacts for dense smoke clouds, while results may perceived realistically for a similar setup with diffuse smoke. In absence of solid perceptional studies, it is therefore best to be clear about the limits and its violations of the model used. Below, some common pitfalls and limits of turbulence methods are listed.

The Scales of Turbulence Turbulence modeling and synthesis base on the fact that turbulence can be separated from the mean flow, and has a uniform dynamic that is well-described by statistical properties. This is true only under certain conditions. Most importantly, turbulence should only be generated for the inertial subrange. Larger scales show nontrivial interaction with flow obstacles, and both forward and backward scattering. This means not only is the K41 energy distribution not valid in this regime, but the dynamics are dominated by coherent structures that can hardly be captured by a statistical model. Here, a simulation is essential as synthesized turbulence will inevitably introduce an unnatural look. This means that the division between simulation resolution and generated subgrid detail has to be carefully chosen. Another aspect of this is that turbulence models base their prediction on the mean flow. This means the predicted turbulence intensity will change depending on the base simulation resolution. This is especially critical if many turbulent details are already resolved by the base solver, as the details will act as turbulence sources, resulting in a strong overprediction of turbulence. In these cases, a full RANS simulation, or an averaged flow field should be used instead of the base simulation.

Isotropy Both the turbulence models and synthesis algorithms presented in this chapter only take into account isotropic turbulence. This is a good assumption for fully developed turbulent flows, but not for areas of turbulence generation or transition. This is due to the fact that turbulence generated from shear will create whirls with a preferred direction, which will only become isotropic over time (see Section 2.2.2). In flows where these areas are clearly visible, such as open channels and turbulent dense smoke, isotropic models should not be used – isotropic turbulence will be perceived as noise disturbing the flow. To include the effects of anisotropy, extensions to both turbulence prediction and synthesis have to be made. Such a model is presented in Section 4.2. However, even with anisotropic methods, such as Reynolds stress transport models, there is no guarantee that turbulence transition is well represented. Turbulence induced by breakdown for example from large coherent structures can hardly be represented using a statistical approach – for this, the breakdown has to be modeled explicitly with methods such as [PTG12], which we present in Section 4.4.

Accuracy of Synthesis Synthesis methods have to fulfill several constraints, which are often incompatible, as discussed in Section 2.4.1. Therefore inevitably compromise solutions have to be implemented, whose effectiveness will depend on a good choice of parameters. Synthesis based on curl noise is especially problematic at steep interfaces of turbulence intensity, e.g. buoyant smoke with a sharp density gradient, and flows with strong strain effects. The latter will induce either strong deformations of the turbulence field, or interpolation artifacts associated with frequent resets. Either way, coherent whirls may be reduced to structure-less noise.

Chapter 3

Literature

Methods for modeling and simulating fluid system have a long tradition in the fields of engineering and physics, and have become vital tools in Computer Graphics, too. The requirements for applications in Computer Graphics are however very different from their counterparts in CFD. Therefore, while similar in theoretical background, methods in Graphics often approach the problems at hand from a different angle, and much is to be learned from studying both sides. In this chapter we will summarize the history and recent works from both Computer Graphics and CFD, focusing on methods for simulating turbulent fluid systems. A good overview can also be found in the textbooks by Wilcox [Wil93] and Pope [Pop00] for turbulence theory, and the recapitulation of fluid simulation methods in Computer Graphics by Bridson [Bri08].

3.1 History

Fluid dynamics are described by the Navier-Stokes equations, a set of partial differential equations. To solve these equations numerically, several discretization schemes have been proposed. Most applications in Engineering base on either finite elements (FEM) [OW72] or finite difference (FDM) / finite volume (FVM) [Hsu81] discretizations. Finite difference methods operate on structured grids, while finite element methods evaluate base functions on irregular meshes. The latter allows to focus resolution on critical regions if known in advance, but require a separate meshing step and have a larger computational overhead. Marker-and-Cell (MAC) discretizations [HW66] extend the FDM approach by placing velocity information on the cell faces instead of centers, which increases precision when calculating derivatives.

Although most methods for numerical simulation of fluid systems use a direct discretization of the Navier-Stokes equations, different principles have been proposed in literature. The fully Lagrangian Smoothed Particle Hydrodynamics (SPH) models fluid dynamics by the interaction of particles with a compressible kernel [GM77]. This method has become popular in Computer Graphics for free-surface problems [MSKG05]. It does however require small timesteps for stability in complex scenarios. Lattice Boltzmann methods on the other hand base on Boltzmann gas dynamics [HPP76], and directly model the flux and collision of fluxes on a regular mesh. While not common in Graphics, they have successfully been used for the simulation of liquids [TIR06].

3.2 Fluid Simulation in Computer Graphics

Fluid simulation in Computer Graphics was popularized by Stam [Sta99], who introduced the combination of semi-Lagrangian advection with first order pressure projection using a MAC discretization. This unconditionally stable, albeit very dissipative type of solver is the most commonly used simulation technique in Computer Graphics and it will serve as reference and base solver in this thesis as well.

Over the years, many extensions of this basic solver have been made. This includes for example methods to simulate liquids [EFFM02], bubble flows [HK03], viscoelastic fluids [GBO04], or interactions with rigid bodies [CMT04]. Possibly the biggest challenge for fluid simulation for Computer Graphics, however, is to represent highly detailed flows efficiently. This problem is two-fold, and due to the dissipative nature of stable semi-Lagrangian advection, and the more fundamental issue of the memory and computation costs involved in high-resolution simulations. Below, recent methods to alleviate issues concerning dissipation, grid resolution and efficiency are summarized.

3.2.1 Low-dissipative Methods

The first part of the problem in preserving detail is the inherent damping of turbulence detail due to numerical dissipation. A popular approach to alleviate this problem is the use of higher order advection schemes. Back and Forth Error Correction [KLLR05], MacCormack advection [SFK+08], QUICK [MCPN08] and CIP methods [KySK08] improve the accuracy of the semi-Lagrangian advection to obtain second- or third order accurateness. The PIC/FLIP approach [ZB05] which is popular in industry productions uses an additional particle set for a more accurate advection. Mullen et al. [MCP+09], on the other hand, propose an implicitly energy-preserving velocity integration scheme for tetrahedral meshes.

Alternatively, Fedkiw et al. [FSJ01] advocate detecting and amplifying existing vortices to combat dissipation. This is extended to multilevel confinement by Jang et al. [JKB+10]. Similarly, manually seeded vortex particles can be used to reinforce turbulence vortices and combat dissipation [SRF05].

While these methods help to reduce the numerical dampening, the detail that can be represented is still inherently limited by the underlying grid resolution.

3.2.2 Sub-grid Methods

Another way to combat numerical dissipation is to adaptively refine the simulation grid in critical areas. Losasso et al. [LGF04] use an adaptive octree structure to discretize the simulation grid, while the method by Feldmann [FOK05] operates on unstructured meshes. The combination of two-dimensional and three-dimensional simulations [IGLF06], [CM11] has also successfully been used to represent large bodies of fluids. The computational overhead introduced by the adaptivity however only pays off if the detailed motion is confined to only a small part of the simulation space, or for very high resolutions.

A different approach to obtain sub-grid accurate results is to track the visible quantity, e.g. smoke or liquid using Lagrangian markers. Traditionally, these fields are represented using Volume-of-Fluid [HN81] methods in Engineering and density fields or level-sets in Graphics. For liquids, particle level-sets [EFFM02] increase the resolution of a level-set

using Lagrangian markers. Bargteil et al. [BGOS06a] and Wojtan et al.[WTGT10] use a triangle mesh to represent and track liquid-air interfaces. Brochu et al. [BBB10] use Voronoi diagrams to generate a surface sub-grid accurate meshes for liquids. A triangle mesh representation to represent the smoke/air surface for plumes has been proposed by Brochu et al. [BB09]. Particle representations are another popular choice, but large numbers are usually necessary to represent dense surfaces without noise. While the Lagrangian markers in these methods allow for the detailed representation below grid scale, the dynamics are still limited by the grid resolution of velocity field. Similar in spirit to the approaches presented in this thesis, Thuerey et al. [TWGT10b] take advantage of the Lagrangian representation and compute sub-grid surface tension dynamics directly on a air/liquid interface mesh.

A major source of turbulent detail is the interaction of fluids with solids. By improving the accuracy of boundary dynamics or modeling the interaction, higher detail levels can be achieved for the fluid system. Two-way coupling of fluids has been addressed by Carlson et al. [CMT04], Guendelmann et al. [GBF03] and Klingner [KFCO06]. More recently researchers have modeled subgrid interactions with objects more accurately through the use of apertures [BBB07, RMSG$^+$08]. Nevertheless, very little previous work in graphics addresses the problem that the thin turbulent boundary layer is not resolved in the simulation, resulting in turbulence not being shed.

3.3 Lagrangian Vortex Methods

Instead of solving the Navier-Stokes equations in the velocity space, they can also be solved in vorticity space. Vorticity describes flow rotation, and is an equivalent description of fluid motion. The vorticity equation can be evaluated on regular meshes and grid using FDM or FEM formulations with similar accuracy and performance as the Navier-Stokes equation. Boundary conditions, obstacle interaction and free surfaces are however more efficiently treated in a velocity representation. Purely Eulerian vortex methods are therefore rarely used in practice. On the other hand, vortex methods have the desirable property that rotational flow features such as eddies are more compactly represented than in the velocity formulation. This makes Lagrangian or hybrid vortex methods an attractive choice for strongly rotational flows, especially those with strong turbulence generation. In this section, an overview of Lagrangian vortex methods in CFD is given.

Lagrangian Primitives The most general and commonly used primitive for vortex methods is the *Vorton*, a point representation of vorticity. Vorton methods in two dimensions have been discussed as early as 1931 [Ros31]. In three dimensions, vortex dynamics are much more involved as a vortex stretching term appears, which is hard to stabilize in a Lagrangian setting. The first three-dimensional methods appeared therefore much later [BM82] with the convergence being proven by Hald [Hal79]. A stable treatment of the vortex stretching term still remains one of the main challenges of vorton methods. To this end, hybrid methods employing an additional grid representation [MG96] have been successfully used to stabilize this term. Vortex filament methods [Leo75, Leo80], which discretize vorticity using one-dimensional space curves, do not share this problem as the vortex stretching term vanishes in their motion equations. However, this comes at the price of ever-increasing geometry due to re-meshing of the connected elements [Cho81]. Although global

re-meshing using vorticity transfer has been proposed to partly reduce geometric complexity [LK01] this is still a largely unsolved problem which restricts simulation run-length and complexity. Finally, vortex sheets, a representation of vorticity on two-dimensional surface, have successfully been used to simulate vorticity dynamics at interfaces. While early methods used points or filaments to discretize the vortex sheet surface [AM89], more recent methods employ a mesh of triangular [BLP98] or quadrangular surface elements [LGOD98, SDT08]. Vortex sheets share the advantages and drawbacks of filaments, but their two-dimensional connectivity makes them very suitable for interface transition modeling, while filaments are often employed for problems including vortex ring breakup.

Extensions A common problem of all Lagrangian vortex methods is the handling of diffusion. Two types of solutions have been proposed, firstly core-spreading [Leo80], in which the primitives kernel is widened, and second vorticity redistribution between neighboring primitives [Gha03]. While these methods have been used successfully for densely sampled vorton simulations, they are largely unsuitable for sparse simulations and the higher dimensional filaments and vortex sheets. Therefore, Lagrangian vortex methods are often applied to problems where diffusion is not relevant and can be omitted, e.g. turbulent flows.

Another big issue is the complexity of the velocity evaluation. Classically, the velocity is obtained by applying the Biot-Savart law between primitives, resulting in a complexity of $\mathcal{O}(n^2)$. Treecodes [PWS+02, Wan04] employ spatial acceleration structures to increase performance, while fast multi-pole methods [WSW+96, Deh02] use far-field approximations of the Biot-Savart law to compute distant interactions more efficiently. Another approach is the Vortex-in-Cell (VIC) method [CK99, CP03, SDT08], which projects vorticity to a grid, and solves a Poisson problem to obtain velocity. This however induces a strong regularization for moderate grid resolutions, which may not be desired.

Finally, extensions to vorticity generation have been proposed. While vorticity generation at obstacles may be represented using boundary conditions, baroclinic generation has to be included as a source term. This theory was proposed by Meng [Men78] and studied by Tryggvason et al. [TA83].

Sparse Sampling In Computer Graphics, vortex methods are mainly used as a sparse representation. Instead of completely discretizing the motion field, vortex primitives are often used to reinforce eddies, and augment a base simulation. Selle et al. [SRF05] uses vortons to reinforce bulk turbulence on a Eulerian base simulation. Filament methods, on the other hand, have been used to control a fluid simulation by manually adding vortex rings [AN05, ANSN06]. Vortex sheet theory is used by Kim et al. [KSK09a] to reinforce the breakup of liquid sheets. However, they discretize the vortex sheet on the grid, and synthesize motion using Eulerian vorticity confinement. While vortex methods are increasingly being used in Graphics, the research base is still very weak, compared to both velocity methods and vortex methods in CFD. Most application in Graphics simply add vortons manually to obtain a more turbulent look and feel.

3.4 Turbulence Methods

The high numerical cost involved in the simulation of small-scale turbulence has given rise to turbulence modeling approaches. These models separate turbulent fluctuations from the mean flow, and aim to describe these fluctuations in a statistical manner, based on properties obtained from the large-scale mean flow. Turbulence modeling methods have a long tradition in Engineering. They are usually employed to estimate the influence of turbulence on the mean flow, e.g. the forces exerted on airplane wings, or the flux reduction in pipe flow. In Computer Graphics, on the other hand, the main interest is not obtaining corrected mean flows, but to obtain the transient turbulent detail itself to enhance visual detail. Therefore, turbulence synthesis methods are employed to generate artificial detail. Advanced synthesis methods use turbulence modeling methods as an estimator to predict the turbulence distribution to synthesize. Below, an overview of turbulence modeling and turbulence synthesis models is given.

3.4.1 Turbulence Modeling

Classical turbulence models in CFD model turbulent viscosity, that is the virtual diffusion turbulence induces on the mean flow [Pra45]. The simplest classical turbulence models which are able to describe non-trivial flows are mixing length models [Sma63], [BL78]. Mixing length models have been used successfully to describe boundary layer flow in e.g. the aerospace industry. However, they rely on manual specification of the mixing length, which is scene-dependent and only known for certain types of flows. To alleviate this issue, complete convective-diffuse models have been proposed, which model the dynamic of turbulence using a set of differential equations without the need for a scene-dependent specification. The Spalart-Almaras models [SA94], which is used in aeronautic applications, directly models turbulent viscosity using one PDE. Two-equation models, on the other hand, model the evolution of parameters such as turbulent kinetic energy and dissipation using two PDEs. The k–ε model [LS74] and the k–ω model [Wil93] are still the most commonly used turbulence models in Engineering today, and included in most CFD applications due to their generality and simplicity. However, the prediction quality and stability of these models strongly depends on the type of flow. Therefore, extensions have been proposed for e.g. better stability in wall-regions [JC94], to cite one example.

The class of classical turbulence models has however two fundamental limits. First, it relies on Reynolds averaging to separate mean flow and turbulence, which might not be sufficient, especially to describe turbulence transition and coherent eddies. To this end, Large Eddy Simulations (LES) have been introduced [Sma63]. LES models use more accurate frequency filters for separation, and directly simulate coherent eddies. Originally, LES was used to describe internal flows in meteorology, but has been adapted for more general applications in CFD as well [HJ00]. A recent extension to LES are Detached Eddy Simulations (DES) [Spa09] which are even more suitable to represent coherent eddies. The second limit concerns the information provided by classical models. While turbulent viscosity is sufficient to describe the virtual diffusion of the mean flow, is does not provide information on flow anisotropy and energy exchange. Reynold stress transport models solve this by modeling not only turbulent viscosity, but the complete turbulent stress tensor [LRR75], [CK95]. Even further, probability density functions such as the General Langevin Equation [Pop83]

directly describe higher-order statistical properties of turbulence using a particle method, while Elliptic Relaxation methods [Dur93] replace the local convection-diffusion processes in turbulence models by global optimization. While all these methods have higher prediction power than classical turbulence models in theory, they also come at the price of higher complexity, numerical cost and tuning effort. Therefore, they are mainly used in the context of CFD research and high fidelity simulation in e.g. meteorology. With the increasing computation power in the recent years, especially LES and Reynolds stress transport models are however slowly becoming more popular for standard applications.

While the fundamentals of CFD turbulence modeling were developed in the 1970s and 80s, this does not imply no progress has been made in recent years. The topics have however shifted from general purpose turbulence modeling towards more specific topics, as it is common in mature fields. Recent topics include the modeling of turbulence in compressive flows [Aup04] or turbulence transition modeling [LMLS06], [DGS07], to cite a few examples. A very interesting recent trend is the development of multi-scale turbulence modeling techniques using LES [CS07] or scale adaptive simulation (SAS) [MK05].

3.4.2 Turbulence Synthesis

Early turbulence methods in Computer Graphics used uniform fields of synthetic turbulence to augmented or replaced basic fluid simulation with synthetic turbulence. Stam [SF93] introduced a method that used a Kolmogorov spectrum to produce procedural divergence free turbulence. This approach was used to model nuclear explosions and flames [RNGF03, LF02]. Bridson et al. [BHN07] suggest taking the curl of vector noise fields to produce divergence free velocity fields. They explicitly address computing flows around objects efficiently by modulating the potential field. These methods however do not take into account the spatial and temporal distribution and dynamics of turbulence, and may lead to unrealistic results for complex flows.

Recent methods use a more complex estimation of flow statistics to better capture the characteristics of turbulence. Kim et al. [KTJG08] use wavelet decomposition to determine local turbulence intensities, and synthesize turbulence using frequency-matched curl noise. This approach assumes that the base solver can resolve the turbulence dynamics. Schechter [SB08b], Narain [NSCL08], Pfaff [PTSG09] use transport models to derive turbulence parameters. This improves the turbulence dynamics, but they need to significantly simplify the energy transport or make assumptions on the flow regime, such as mixing-length models to close their one-equation energy model. Often, complete two-equation models [PTC+10] are more general and easier to deal with (Section 4.2).

3.5 Recent works

Turbulence methods and vorticity representations for fluid simulations are an active area of research in Computer Graphics. It is therefore not surprising that a number of works on these topics has been published concurrent to or as a follow-up work to the methods presented in this thesis. These approaches will be discussed below.

Turbulence Synthesis While this thesis focusses on synthesis based on curl-noise textures and vorticity, alternative methods for synthesis have been presented in recent years. Chen et al. [CZY11] present a hybrid turbulence method, which synthesizes turbulence on a particle system, based on Section 4.2. Instead of using a RANS-based turbulence model, however, they follow a pdf approach and solve the Langevin equation [Pop00] on the particle system. The resulting velocity update is directly used as detail motion of the particles. This avoids both the coherence problem of curl-noise synthesis and the stability issues of two-equation turbulence models employed in Section 4.2. On the other hand, by interpreting the stochastical Langevin velocities as actual particle motion, the synthesized motion will not be divergence-free and spatially coherent.

Zhao et al. [ZYC10], on the other hand, propose the use of random forcing on an upsampled grid as a synthesis method. The divergence-free force fields are precalculated and follow a given energy spectrum. This representation does not require a separate set of markers such as vortex primitives or texture coordinates for curl-noise synthesis. The synthesized forces can not directly be animated; instead, alternating pre-computed fields are applied. While this temporal inconsistency may lead to artifacts and flow disturbance if the method is used beyond small scales, this method might provide a very attractive alternative for synthesis, if extended such that force fields can be updated at runtime.

Vorticity methods The original vortex particles paper [SRF05] was extended by Yoon et al. [YKH+09] and [PTSG09]. Both methods apply the vortex particles onto an upsampled higher resolution grid. The later method also includes a turbulence preditor for wall turbulence as described in Section 4.2. Kim et al. [KLySK12] extended the vortex particle approach for buoyant sources. The source terms are calculated on the grid and then mapped to the vortex particle system.

Weissmann et al. [WP10] proposed a simulation driven entirely by vortex filaments, with source terms for obstacle-induced shedding. They are able to represent flows with a medium level of turbulence in a very compact fashion, which makes the method very efficient in this regime. This approach was recently extended by an improved re-sampling scheme for filaments, which limits the geometry growth [BP12]. An introduction to the vortex primitives of particles, filaments and sheets and a comparison of their strengths and weaknesses can be found in Section 4.4.1.

Vortex sheets [PTG12, BKB12] as described in Section 4.4 are a good choice to represent turbulence from buoyant plumes. The main difference between the two methods is the acceleration of vorticity integration. Pfaff et al. use a local evaluation scheme while Brocho make use of a fast multipole method (FMM). The later has the advantage that no Eulerian base simulation is necessary, which obviates the need to tune the grid's buoyancy terms to the mesh-based buoyancy and avoids artifacts induced by the scale separation. On the other hand, it is hard to represent inflows, source terms, obstacle interaction and the coupling to e.g. turbulence methods in this pure vorticity formulation, which are better handled using the first method. It therefore depends on the application scenario which of the methods is best suited.

Chapter 4

Adding Turbulent Detail

In the following sections we will outline several approaches for modeling turbulent effects. *Wavelet Turbulence* is a very practical approach that outlines a practical way to apply small scale detail for coarse fluid simulations. Afterwards, we will explain several approaches to increase physical accuracy, for anisotropic effects, near obstacles and for buoyancy effects.

4.1 Wavelet Turbulence

For visual simulation of fluids scalability and user interaction are significant problems. Most recent methods directly solve the Navier-Stokes or incompressible Euler equations over a mesh. However, if features smaller than a mesh element are required, as is common when simulating large-scale phenomena such as explosions or volcanic eruptions, the mesh must be refined in some way. This results in a linear increase in memory use and a greater than linear increase in the running time. elements. Adaptive methods [KFCO06] [LGF04] have been developed that only add elements near regions of interest, but the overall asymptotic complexity remains the same, and the implementation complexity increases.

We instead propose an algorithm that generates small-scale fluid detail procedurally. We use a wavelet decomposition to detect where small-scale detail is being lost, and apply a novel incompressible turbulence function to reintroduce these details. Instead of enforcing the Navier-Stokes equations over all spatial scales, we enforce Navier-Stokes over low frequencies and Kolmogorov's turbulence spectrum over high frequencies. We have eliminated the global dependencies of a linear solver over the grid, so the high-resolution portion of the algorithm parallelizes trivially. The derivation of our method additionally provides insight into the popular Perlin `turbulence()` function [Per85] and suggests a physical basis for its success. The algorithm requires only the velocity field of an existing fluid simulation as input, and can be run as a fully decoupled post-processing step. This allows users to rapidly iterate on fluid simulation designs until the desired overall behavior is achieved, then later add small-scale details prior to rendering. The fluid solvers commonly used in graphics do not allow this. Increasing the resolution changes the effective viscosity of the fluid, produces significantly different motions at a higher resolution, and invalidates much of the design work done on lower resolutions. In contrast, our method introduces new energies in a band-limited manner that guarantees existing structures are preserved.

Our contributions are as follows:

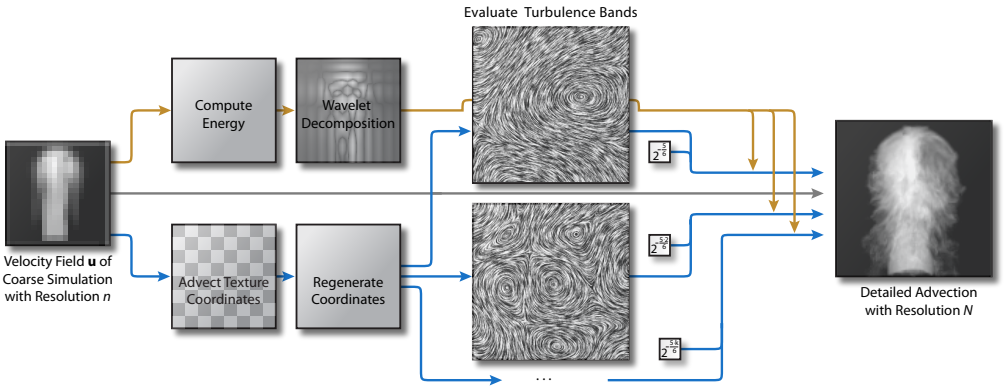


Figure 4.1: **Overview:** A low-resolution n^3 velocity field $\mathbf{u}$ is used to synthesize a high-resolution N^3 density field. Procedural turbulence is added according to the wavelet decomposition of the energy, and the resulting eddies are advected via texture coordinates until they scatter.

- An incompressible turbulence function that can generate arbitrary energy spectra.
- A method for estimating the small-scale turbulence that is lost by a simulation, and re-synthesizing it in a way consistent with Kolmogorov theory.
- A method of preserving the temporal coherence of the synthesized turbulence.
- A large- and small-scale fluid detail decoupling that allows the latter to be edited independently.

4.1.1 Procedural Wavelet Turbulence

In this section we describe how to efficiently construct an incompressible turbulence function. We adopt the following notation for the remainder of this paper. Bold denotes a vector, and non-bold denotes a scalar. The special variable $\mathbf{x}$ denotes a spatial position, k denotes a spectral band, and $\mathbf{u}$ denotes a velocity field. A carat denotes a wavelet transform, so $\widehat{\mathbf{u}}(\mathbf{x},k)$ denotes the spectral component of velocity field $\mathbf{u}$ at position $\mathbf{x}$ in spectral band k. Additionally, n always refers to the grid resolution, n^3, of $\mathbf{u}$, and $\mathbf{v_x}$, $\mathbf{v_y}$, $\mathbf{v_z}$ refer to the Cartesian unit vectors, ie $\mathbf{v_x} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$.

Incompressible, Band-Limited Noise: Wavelet Noise [CD05] was developed as a replacement for Perlin Noise [Per85]. The noise is guaranteed to exist only over a narrow spectral band, which makes more sophisticated filtering possible. We will use this band-limited property as a first step in constructing an incompressible turbulence function.

The Wavelet Noise function ω is a scalar function, whereas we are interested in vector fields. As was recently demonstrated for graphics [BHN07], a calculus identity can be used to construct a divergence-free vector field by taking the curl of a scalar field. We can respectively construct 2D and 3D vector fields using ω:

$$\mathbf{w}_{2D}(\mathbf{x}) = \left(\frac{\partial \omega}{\partial y}, -\frac{\partial \omega}{\partial x} \right) \tag{4.1}$$

$$\mathbf{w}(\mathbf{x}) = \left(\frac{\partial \omega_1}{\partial y} - \frac{\partial \omega_2}{\partial z}, \frac{\partial \omega_3}{\partial z} - \frac{\partial \omega_1}{\partial x}, \frac{\partial \omega_2}{\partial x} - \frac{\partial \omega_3}{\partial y} \right) \tag{4.2}$$

We primarily use the 3D case (Eqn. 4.2) unless otherwise stated. The 3D case requires three different noise tiles, which we have denoted ω_1, ω_2 and ω_3, but in practice we use offsets into the same noise tile. The derivatives can be evaluated directly because ω uses B-spline interpolation. Instead of the usual quadratic B-spline weights, $\left[\frac{t^2}{2}, \frac{1}{2}+t(1-t), \frac{(1-t)^2}{2}\right]$, we apply the derivative weights, $[-t, 2t-1, 1-t]$, in the desired directions. The resultant vector field is guaranteed to be incompressible. The field retains the same band-limited properties of the original signal because differentiation is a linear high-pass filter in the frequency domain, which does not add new frequencies by definition. This is sufficient to ensure that the 2D vector field is band limited. Additionally, we observe that adding two signals together is also linear, so the 3D case is also band limited.

From a visual perspective, $\mathbf{w}$ generates a vector field of randomly distributed, tightly packed eddies of fixed size. Two bands of the function can be seen in Figure 4.1. Next, we will use $\mathbf{w}$ to generate a vector field according to Kolmogorov's theory of turbulence.

Kolmogorov Wavelet Turbulence: Kolmogorov famously showed that, for a homogeneous inviscid fluid, while the local structure of its velocity field may be perpetually chaotic, the global energy spectrum approaches an equilibrium state that can be described in very simple terms [Fri95]. The energy density e of some grid cell $\mathbf{x}$ is its kinetic energy,

$$e(\mathbf{x}) = \frac{1}{2}|\mathbf{u}(\mathbf{x})|^2. \tag{4.3}$$

The total energy e_t of a grid is then the sum over all grid cells. Kolmogorov's theory deals with the frequency spectrum of e_t. While the original theory used the Fourier transform, we use a wavelet transform because it provides both spatial and frequency information. If we compute e_t for each band k of $\hat{\mathbf{u}}(\mathbf{x},k)$ we obtain an energy spectrum $e_t(k)$. One of the key results of Kolmogorov theory is that the energy spectrum of a turbulent fluid approaches a five-thirds power distribution:

$$e_t(k) = C\varepsilon^{\frac{2}{3}}k^{-\frac{5}{3}}. \tag{4.4}$$

Where C and ε are the Kolmogorov constant and the mean energy dissipation rate per unit mass. The $-\frac{5}{3}$ scaling exponent holds for both Fourier and wavelet spectra [PPB95], and the substitution is common in fluid dynamics [FKPG96].

Using our noise function $\mathbf{w}$, we can procedurally construct a velocity field that produces this power distribution. We first observe that (4.4) can be rewritten as the recurrence relation:

$$e_t(2k) = e_t(k)2^{-\frac{5}{3}}, \quad e_t(1) = C\varepsilon^{\frac{2}{3}}.$$

The velocities can then be stated analogously using (4.3):

$$|\widehat{\mathbf{u}}(\mathbf{x},2k)| = |\widehat{\mathbf{u}}(\mathbf{x},k)|2^{-\frac{5}{6}}, \quad |\widehat{\mathbf{u}}(\mathbf{x},1)| = 2^{\frac{1}{2}}C^{\frac{1}{2}}\varepsilon^{\frac{2}{6}}. \tag{4.5}$$

Since $\mathbf{w}(\mathbf{x})$ is band limited, it can be substituted in for $\widehat{\mathbf{u}}(\mathbf{x},k)$. Our final wavelet turbulence function is then a series version of (4.5):

$$\boxed{\mathbf{y}(\mathbf{x}) = \sum_{i=i_{\min}}^{i_{\max}} \mathbf{w}(2^i\mathbf{x})2^{-\frac{5}{6}(i-i_{\min})}.} \tag{4.6}$$

The variables $[i_{\min}, i_{\max}]$ can be used to control the spectral bands that $\mathbf{y}(\mathbf{x})$ applies to.

Discussion: Eqn. 4.6 shares much of the appeal of Perlin's widely used `turbulence()` function [Per85]. Perlin's `turbulence()` also sums successive bands of a noise function $p(\mathbf{x})$ to obtain a "visually turbulent" scalar function. It can be stated in terms very similar to ours:

$$\texttt{turbulence}(\mathbf{x}) = \sum_{i=i_{\min}}^{i_{\max}} p(2^i\mathbf{x})\frac{1}{2}^{i-i_{\min}}. \tag{4.7}$$

Our function is essentially a *vector* version of (4.7) that is also band-limited, guarantees incompressibility, and produces the Kolmogorov power distribution. Interestingly, $2^{-\frac{5}{6}} \approx 0.56$, which is close to the heuristic $\frac{1}{2}$ value used by Perlin. If scalar wavelet noise w were used instead of p, and $2^{-\frac{5}{6}}$ instead of $\frac{1}{2}$, a Kolmogorov-Obukhov-Corrsin *scalar turbulence spectrum* would be obtained [SS00], which suggests a physical reason for Perlin's success at generating visually turbulent textures.

4.1.2 High-Resolution Fluid Synthesis

We now show how to use the turbulence function $\mathbf{y}(\mathbf{x})$ to add new high-frequency components to $\mathbf{u}$. First, we motivate our approach by discussing the intuition that underlies Kolmogorov's theory.

Background: Physically, the five-thirds distribution occurs because of *scattering* [Fri95]. As an eddy is advected by an incompressible field, it is stretched in one direction and compressed in another. Eventually these deformations break the eddy into two eddies of half the size. This process is called *forward scattering*. The opposite phenomena, *back scattering*, occurs when smaller eddies combine to form larger ones, but forward scattering usually dominates. Kolmogorov's five-thirds spectrum describes the energy distribution after sufficient time has passed that eddies injected at a fixed scale (the *integral scale*) have forward scattered into higher frequencies. At much higher frequencies, viscosity becomes dominant, so at a second scale (the *ultraviolet cutoff*) the energies start to dissipate at a much faster rate than the five-thirds exponent. In graphics, dissipation damps out interesting fluid detail, so the viscous term is usually dropped from the Navier-Stokes equations. Theoretically this places the ultraviolet cutoff at the Nyquist limit, $\frac{n}{2}$ for an n^3 mesh. As mentioned previously, significant dissipation still occurs before the Nyquist limit, and many techniques have been developed to address this issue. We instead focus on the separate problem of placing the ultraviolet cutoff *beyond* the Nyquist limit.

Injecting Turbulence: Our goal is to synthesize a high-resolution N^3 density field D from a low-resolution n^3 velocity field $\mathbf{u}$. We use lowercase to denote variables at the lower n^3 resolution and uppercase for fields at the higher N^3 resolution. We define an interpolation function $\mathscr{I}(\mathbf{u},\mathbf{X})$ that interpolates $\mathbf{u}$ at the high-resolution location $\mathbf{X}$, and $\mathscr{A}(\mathbf{U},D)$ as the advection of D by $\mathbf{U}$.

We will now focus on synthesizing an N^3 velocity field $\mathbf{U}$. The simplest method is interpolation: $\mathbf{U}(\mathbf{X}) = \mathscr{I}(\mathbf{u},\mathbf{X})$. This smooths out the velocities according to the interpolation method, but it does not generate any new eddies in the new $[n, \frac{N}{2}]$ spectral bands. A more sophisticated method is to compute the energy $e_t(\frac{n}{2})$ of the smallest eddies in $\mathbf{u}$, and use it

to weight our turbulence function:

$$\mathbf{U}(\mathbf{X}) = \mathscr{I}(\mathbf{u},\mathbf{X}) + 2^{-\frac{5}{6}} \cdot e_t\left(\frac{n}{2}\right) \cdot \mathbf{y}(\mathbf{X}) \tag{4.8}$$

We only want to inject energy into the new $[n, \frac{N}{2}]$ bands, so we set $i_{\min} = \log_2 n$ and $i_{\max} = \log_2 \frac{N}{2}$ in $\mathbf{y}(\mathbf{X})$. Intuitively, this approach assumes that the energy spectrum of $\mathbf{u}$ follows the Kolmogorov distribution, computes the last resolved value, $e_t(\frac{n}{2})$, and uses it to extrapolate energies over the new $[n, \frac{N}{2}]$ bands.

This method approximates the high-frequency components of $\mathbf{U}$ as fully developed, homogeneous turbulence in a manner similar to Stam and Fiume [SF93]. While this can produce acceptable results, because $\mathbf{y}(\mathbf{X})$ is weighted globally, it can spontaneously produce small-scale eddies in regions where there is no larger-scale precursor. The weighting should instead vary spatially, with turbulence added only when an eddy in $\mathbf{u}$ forward scatters into a previously unresolvable frequency. The resulting turbulence then needs to be advected along with the flow.

Detecting Scattering: Forward scattering can be detected by weighting $\mathbf{y}(\mathbf{X})$ by $\widehat{e}(\mathbf{x}, \frac{n}{2})$ instead of $e_t(\frac{n}{2})$. This locally extrapolates the energy spectrum using the same intuition as Eqn. 4.8. Like Neyret [Ney03], we then advect a set of texture coordinates $\mathbf{c} = (c_u, c_v, c_w)$ along with the flow, and evaluate $\mathbf{y}$ using the advected value. If we detect that the local texture coordinates are causing $\mathbf{y}$ to deviate too far outside of its original spectral band, they are regenerated to the original local values. The method in Neyret [Ney03] uses a heuristic strain criterion to trigger regeneration because it is intended for generic texture advection. We are specifically advecting eddies, so we can use more physically based criteria. We quantify the amount of local deformation using the Jacobian of the texture coordinates, which we denote $\mathbf{J}(\mathbf{c}(\mathbf{x}))$.

For the eigenvalues $\lambda(\mathbf{c}(\mathbf{x}))$ of $\mathbf{J}(\mathbf{c}(\mathbf{x}))$, if $\max(|\lambda(\mathbf{c}(\mathbf{x}))|) \geq 2$, from the physical standpoint, the local eddy has forward scattered into a higher spectral band. If $\min(|\lambda(\mathbf{c}(\mathbf{x}))|) \leq \frac{1}{2}$, then the eddy has back scattered to a lower band. In both cases, the eddy is no longer physically (or visually) coherent, so the texture coordinate should be regenerated.

Texture Distortion: As the advected texture coordinates stretch and rotate, $\mathbf{y}$ does as well. This potentially violates incompressibility because, as the texture coordinates deform, the derivatives $[\partial\omega/\partial c_u,\ \partial\omega/\partial c_v,\ \partial\omega/\partial c_w]$ no longer correspond to derivatives along the Cartesian axes, $[\partial\omega/\partial x,\ \partial\omega/\partial y,\ \partial\omega/\partial z]$. The derivatives can be obtained by projecting the Cartesian axes into texture space and taking the directional derivative:

$$\left[\frac{\partial\omega}{\partial c_u}\frac{\partial\omega}{\partial c_v}\frac{\partial\omega}{\partial c_w}\right]\mathbf{J}(\mathbf{c}(\mathbf{x}))^{-1}\begin{bmatrix}\mathbf{v_x} & \mathbf{v_y} & \mathbf{v_z}\end{bmatrix} = \left[\frac{\partial\omega}{\partial x}\frac{\partial\omega}{\partial y}\frac{\partial\omega}{\partial z}\right]$$

The resulting Cartesian derivatives are then used in (4.2). We denote a modified turbulence function that takes in a texture coordinate and performs this projection as $\mathbf{z}(\mathbf{c})$.

Final Algorithm: Our final equation for generating a high-resolution velocity field is:

$$\mathbf{U}(\mathbf{X}) = \mathscr{I}(\mathbf{u},\mathbf{X}) + 2^{-\frac{5}{6}}\mathscr{I}\left(\widehat{e}\left(\mathbf{x},\frac{n}{2}\right),\mathbf{X}\right)\mathbf{z}(\mathscr{I}(\mathbf{c},\mathbf{X})). \tag{4.9}$$

As before, the values of $i_{\min} = \log_2 n$ and $i_{\max} = \log_2 \frac{N}{2}$ are used to evaluate $\mathbf{z}$ over the appropriate spectral bands. The full algorithm is described below, and illustrated in Figure 4.1:

SYNTHESIZE-FLUID($\mathbf{u}$)
1 $\mathscr{A}(\mathbf{u}, \mathbf{c})$
2 Compute $\widehat{e}(\mathbf{x}, \frac{n}{2})$, $\mathbf{J}(\mathbf{c}(\mathbf{x}))$, $\lambda = \lambda(\mathbf{c}(\mathbf{x}))$
3 **if** $\max(|\lambda|) \geq 2$ or $\min(|\lambda)|) \leq \frac{1}{2}$
4 **then** Regenerate $\mathbf{c}(\mathbf{x})$.
5 Synthesize $\mathbf{U}$ using (4.9)
6 $\mathscr{A}(\mathbf{U}, D)$
7 **return** D

Complexity: Almost all the steps occur on the smaller n^3 grid, with the N^3 grid only being used for the final generation of $\mathbf{U}$ and advection of D. The algorithm requires six additional arrays of size n^3: $\mathbf{c}$, $\min(|\lambda|)$, $\max(|\lambda|)$ and $\widehat{e}$. The values of $\mathbf{J}$ can be discarded once λ is computed. The large N^3 arrays for $\mathbf{U}$ and D are a drawback because they increase memory use to $O(N^3)$. However, if $\mathscr{A}(\mathbf{U}, D)$ is implemented using a semi-Lagrangian scheme, only D is instantiated explicitly because each $\mathbf{U}(\mathbf{X})$ is discarded after computing $D(\mathbf{X})$. If particles are used to track the density, even the explicit D is unnecessary and $O(n^3)$ memory use is achieved.

Obstacles and Control: Obstacles can be incorporated with minor modifications. The main issue is that, if velocities inside an obstacle are set to zero, the discontinuity in $\mathbf{u}$ can cause a jump in $\widehat{e}(\mathbf{x}, \frac{n}{2})$. To prevent this we extrapolate energy values from the obstacle boundary inwards using a fast marching method prior to computing $\widehat{e}(\mathbf{x}, \frac{n}{2})$. In order to give a user control over the turbulence, $\mathbf{y}$ can be weighted by an arbitrary volumetric function $v(\mathbf{X})$ in addition to $\widehat{e}(\mathbf{x}, \frac{n}{2})$. A user can amplify turbulence in regions by setting $v(\mathbf{x}) > 1$ or can suppress detail by specifying $v(\mathbf{x}) < 1$. In either case, the initial $\widehat{e}(\mathbf{x}, \frac{n}{2})$ value provides a good default setting.

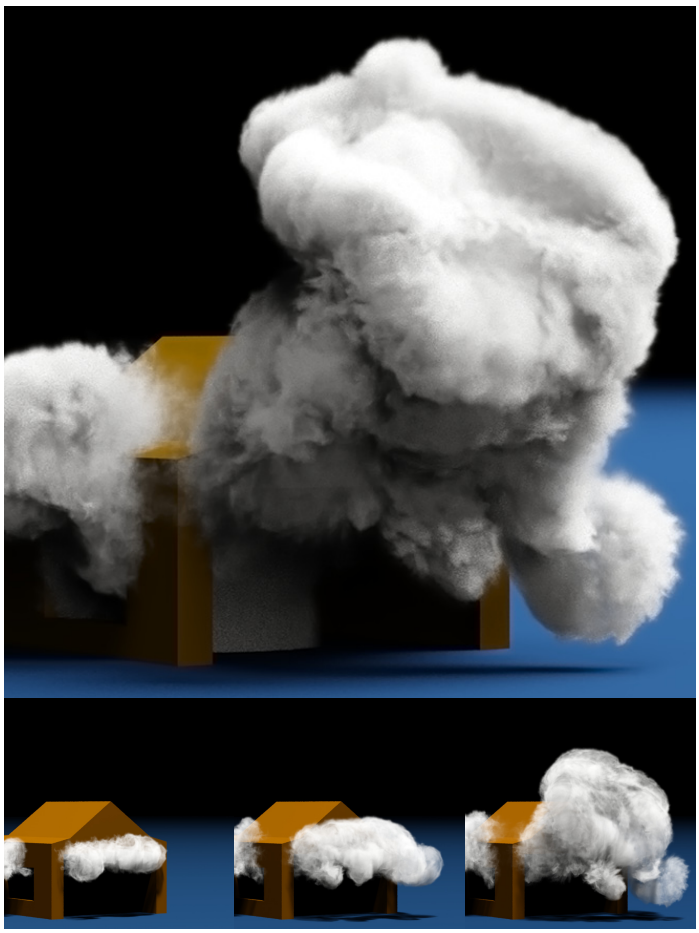


Figure 4.2: Flow around a complex obstacle with grid-based densities. Wavelet turbulence synthesized a $720 \times 576 \times 576$ grid from a $80 \times 64 \times 64$ grid. Each frame took less than two minutes on an eight core workstation.

4.1.3 Results

The $\mathscr{I}$ method in §4.1.2 is ideally the wavelet upsampling method. However, we compared the results of the upsampler with simple linear interpolation and found the difference negligible, and so favored the slightly faster linear interpolation. The $\mathscr{A}$ method we used was the MacCormack method of Selle et al. [SFK+08]. We used a standard fluid solver [FSJ01] for our low- resolution simulations, but added a small amount of heat diffusion to stimulate velocities outside of existing low-resolution densities. We used co-located grids to minimize the number of times (4.9) is evaluated. On a staggered grid, cell-centered averages could be computed and then sent to (4.9) to achieve similar savings.

Figure 4.3 shows a particle simulation of smoke. We are able to achieve a very high effective resolution because a high-resolution grid is unnecessary. In Figures 4.2 and 4.4, we show a simulations of smoke interacting with static obstacles. Complete descriptions are available in the figure captions. Comparisons of our algorithm to Stam and Fiume

Figure 4.3: Flow around a sphere with particle-based densities. The low resolution simulation is shown in the left half of the top image.

[SF93], Bridson et al. [BHN07], and an explicit full-resolution simulation are available in the enclosed video. For the explicit comparison, we downsampled an existing high-resolution simulation and then re-synthesized the discarded bands. Our algorithm appears to resolve more high-frequency detail, as it does not suffer from dissipation near the Nyquist limit. Preliminary timings suggest our method is significantly faster than an equivalent full-resolution simulation. For a 250^3 simulation, the single-threaded version of our method ran roughly seven times faster than the full solver.

All the steps of the complete algorithm only require local support, so the computation parallelizes trivially. We successfully achieved significant speedups by adding a single OpenMP directive to the outermost loop. The main computation, the evaluation of $\mathbf{z}$, runs 3.7 times faster on a four core workstation. This also suggests that the algorithm will perform very well on GPUs.

4.1.4 Conclusions

We have demonstrated a wavelet method that is suitable for adding detail to existing fluid simulations as a user-controlled post-process. While we have demonstrated results using Eulerian grids, all our method requires is the ability to point-sample a velocity field, so it applies to Lagrangian simulations as well.

Figure 4.4: Flow around a sphere with grid-based densities. The low resolution simulation is shown in the left half of the top image. Wavelet turbulence synthesized a 400^3 grid from a 50^3 grid. Each frame took an of average 30 seconds on a four core workstation.

The method has several limitations. By design, it does not reproduce the results of explicit high-resolution simulations. This is partly because back scatter from higher bands into the low-resolution simulation has been explicitly suppressed. A physically based method of estimating back scatter could be developed to address this. Also, the quality of obstacle interaction is totally dependant on the interaction quality at low-resolution. As low-resolution results improve, our results will improve correspondingly. It is also possible for the wavelet transform to introduce very small coefficients into the high-frequency bands, resulting in non-zero turbulence breaking up an otherwise laminar flow. While clamping is a coarse solution, the issue is worth further study.

Several issues were found to be visually unimportant. First, the coordinates **c** should be regenerated on a per-band basis, as higher frequency bands deform faster. Per-band coordinates were implemented and did not make an appreciable visual difference, so they were discarded for simplicity. The more relevant guarantee that the bands of **y** do not impinge on those of **u** remains intact. Second, a direct visualization of **c** shows significant popping because we do not perform any blending during regeneration. However, the popping does not introduce noticeable artifacts in the vector fields because it only occurs after the eddy is no longer visually relevant. Finally, the non-uniform weighting of **z** by $\widehat{e}(\mathbf{x}, \frac{n}{2})$ introduces some compressibility. The solution is to project all the energies into texture space and weight the noise tile by $\widehat{e}(\mathbf{x}, \frac{n}{2})$. We found these extra computations unnecessary, as the error introduced is small compared to the error of the advection method.

The per-band weights of our method can be modified arbitrarily, allowing for 'spectral shaping' in the same way as Perlin noise, and allows many of the procedural texturing tricks used for scalar textures to be applied to the vector regime. The Kolmogorov theory does not account for chemical effects such as those in explosions, but since these effects primarily introduce perturbations to the energy spectrum, they could potentially be captured through shaping. Automatic methods of coupling the weights to existing methods [MTPS04] also have the potential to generate very specific, fine-grained fluid detail.

4.2 Anisotropic Turbulence Modeling

In this chapter, we will have a look at turbulence synthesis and prediction techniques which are able to represent complex features of turbulence without strong reliance on the base solver, and which are suitable for GPGPU computation, based on the paper [PTC+10]. Such approaches are able to generate detailed turbulent simulations with millions of particles at high frame-rates. This makes them good candidates to enhance the realism of fluid simulations in interactive applications, such as Computer Games.

It is generally very difficult to resolve the fine details of turbulent flows in a simulation, and real-time systems impose even stricter limits on the simulation resolution used. Real-time simulations would therefore be a prime field of application for turbulence synthesis methods – as synthesizing turbulent detail is more efficient and scales better than the direct simulation of this detail. However, turbulence prediction using real-time solvers is non-trivial. The low resolution available to the base simulation will result in a coarse and diffusive velocity field which dampens out flow instabilities. This means many types of turbulence will not even be generated in this base simulation. Methods such as Wavelet Turbulence Section 4.1.4 which directly rely on the base grid as a turbulence predictor will

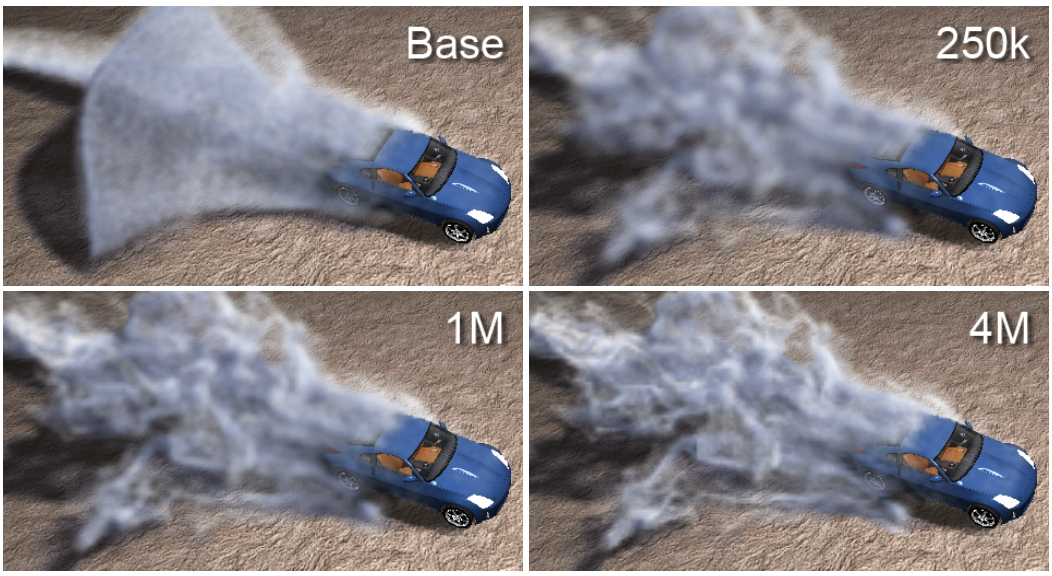


Figure 4.5: Here, the simulation of the wake of a moving car is shown. The base simulation in the top left pictures uses a resolution of only $32 \times 8 \times 32$. This simulation is augmented with the turbulence method presented in this section, with a varying number of particles from 250k to 1M and 4M from left to right. For the simulation with 1M particles we achieve 15 frames per second on average, including rendering. While the amount of detail directly depends on the number of particles used, the overall flow remains consistent.

therefore fail in this scenario. In addition, the real-time constraint precludes the use of high-resolution grids to store and render the generated turbulence.

Therefore, a synthesis method is chosen which uses an energy transfer model to predict turbulence intensity even on low resolution grids. In particular, anisotropy is explicitly modeled, which allows us to represent the anisotropic turbulence formation process using turbulence synthesis, instead of relying on the simulation, which lowers the requirement for base solver resolution. The method is designed to operate directly on the Lagrangian markers used for rendering, and therefore allows us to perform synthesis where needed, without relying on a high-res grid. The key element in this method is the offloading of complexity from the fluid solver to the particle system and turbulence model, so the detail of the simulation can easily by adjusting the number of particles, without changing the large scale behavior.

4.2.1 Overview

To efficiently simulate small scale turbulence it is not feasible to directly represent the turbulent motion using a high resolution velocity grid, as the computational effort increases strongly with grid resolution. Instead, we describe the turbulence field by its statistical properties, and synthesize turbulence only where needed. Our approach is based on a separation of the large scale dynamics from the small scale turbulence: large scales are computed using a low resolution fluid solver, while the turbulent detail with anisotropic effects

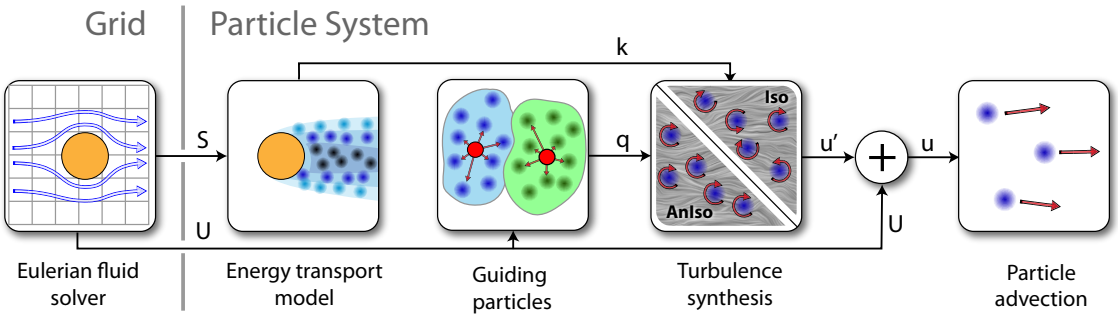


Figure 4.6: An overview of the algorithm. A low resolution grid-based solver provides a base velocity and strain field. For each particle, a turbulence model is computed, which drives the turbulence synthesis with isotropic and anisotropic turbulence. The particles velocity is given by the large scale velocity from the grid and the small scale turbulent velocity.

is computed on the particle system. The particles coincide with the smoke particles used for rendering, while the grid-based solver is used to obtain the large scale characteristics of the flow. The turbulence is computed and synthesized directly on the particles, each of which is influenced by a texture based turbulence representation, and stores a preferred axis of rotation for anisotropic effects. An overview of our model is given in Fig. 4.6.

In Section 4.2.2 we describe the modified energy transport model used for prediction of the spatial distribution of turbulence. This drives the synthesis model Section 4.2.2, which is extended for anisotropy in Section 4.2.2. Section 4.2.3 discusses implementation details. Finally, the complete model is tested in reference scenarios and compared to other methods in Section 4.2.4.

4.2.2 Turbulence Model

Turbulence is described using an energy representation, as this allows us to adapt powerful and proven transport models for our simulations. The spatial and temporal energy distributions driving the turbulence synthesis are obtained using a modified k–ε turbulence model that we will explain in the following sections.

Energy transport

To simulate the energy dynamics, we use a modified version of the k–ε model by Launder and Sharma [LS74]), which is one of the most widely used turbulence models in CFD. It is a complete two-equation model, which means that unlike one-equation models requires no additional problem-dependent assumptions such as mixing length, which are hard to estimate for the general case. On the basis of a large scale flow field $\bar{\mathbf{U}}$, it models the two variables k and ε on an averaged large scale. While k represents the turbulent kinetic energy contained in the smaller scales, ε stands for the dissipation of the turbulence structures. The k–ε model and other turbulence models are discussed in the theory section Section 2.2.1.

We start with the partial differential modeling equations of the k–ε model

$$\frac{\partial k}{\partial t} + \bar{\mathbf{U}}\nabla k = \nabla\left(\frac{\nu_T}{\sigma_1}\nabla k\right) + P - \varepsilon \tag{4.10}$$

$$\frac{\partial \varepsilon}{\partial t} + \bar{\mathbf{U}}\nabla \varepsilon = \nabla\left(\frac{\nu_T}{\sigma_2}\nabla \varepsilon\right) + \frac{\varepsilon}{k}\left(C_1 P - C_2 \varepsilon\right) . \tag{4.11}$$

Both equations share the same structure: the left-hand side contains an advection in the mean flow field. The right-hand side of both equations consist of a viscous diffusion term, a production and a dissipation term, in that order. The equations are coupled to the mean flow field via the velocity field $\bar{\mathbf{U}}$ and the mean flow strain S_{ij} in the production term

$$P = 2\nu_T \sum_{ij} {S_{ij}}^2 . \tag{4.12}$$

Instead of discretizing and solving these PDEs on a Eulerian grid, we compute them directly on the particle system. This means each particle stores a value of k and ε, while the coupling parameters of mean flow velocity and mean flow strain are interpolated from the base flow field. In this Lagrangian setting, (4.10) and (4.11) simplify, as the advection is inherently handled by the motion of the particles with the flow. Next, let us reconsider the role of the diffusion term. The turbulent viscosity ν_T is a virtual diffusion, which models the averaged effect of the small-scale turbulent motion as a viscous diffusive effect on the larger scale of the model. In contrast to CFD however, we solve the model equations and track the turbulence properties on the particle set. The particle set is not only advected by the averaged large-scale flow, but also contains a secondary advection by the synthesized detail motion. This small-scale advection term causes turbulent mixing of the particles and thus implicitly describes the behavior which is modeled by turbulent diffusion term for the large scales in the standard k–ε model. Therefore, the turbulent viscosity term vanishes in our representation. Avoiding these terms also allows us to track k and ε independently for all particles and skip a costly communication step, which is important for GPGPU parallelization. So far, the isotropic version of our model therefore consists of the following equations

$$\frac{\mathrm{D}k}{\mathrm{D}t} = P - \varepsilon \tag{4.13}$$

$$\frac{\mathrm{D}\varepsilon}{\mathrm{D}t} = \frac{\varepsilon}{k}(C_1 P - C_2 \varepsilon). \tag{4.14}$$

Turbulence Synthesis

To obtain the detailed motion for the particle system, in addition to the base flow velocity, a detail velocity component is evaluated directly on the particles. Instead of using the base solver to predict the turbulence distribution (as in the previous chapter), we will now use the energy transfer model described above to estimate the spatial distribution of turbulence. First, let us discuss the energy model spectrum, and the new approach to compute isotropic turbulence. Extensions for anisotropic effects will be presented in the following chapter.

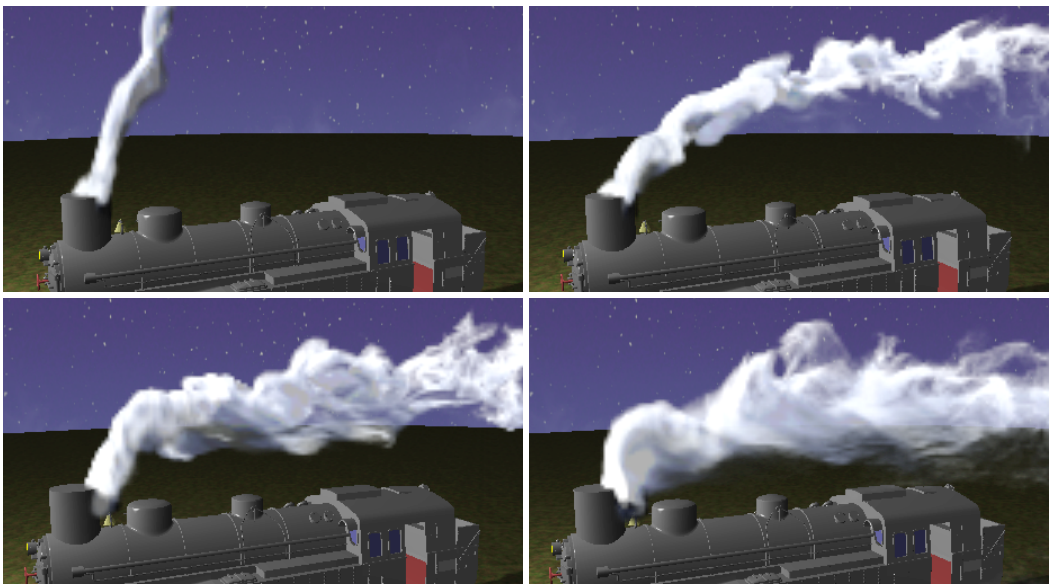


Figure 4.7: We apply the method to an accelerating train that, in the end, comes to an abrupt halt. Due to the energy transport model, turbulence intensities correctly adapt to the direction of the flow and the train's velocity.

Model Spectrum Turbulent energy is usually studied with respect to the spatial scales of the structures in the flow field. The production of turbulent energy is typically concentrated in the energy-containing range of a fluid, its large scales, while dissipation to heat is growing stronger for the small scales. In between these two extrema lies the so called inertial subrange, in which the predominant energy transport mechanism is forward-scattering, transporting the energy from large to small scales. This transfer process can be modeled with time dependence, e.g. using the transient model of Obukhov [Obu41]. However, the model is often not practical, as the exponential nature of the transfer terms requires a high spectral and temporal resolution for a stable solution [Pan71]. Fortunately the transfer phenomena in a flow quickly drive the distribution to a stationary solution for fully-developed turbulence. Therefore, practical approaches typically focus on the stationary solution only. Our method concentrates on scales mostly within the inertial subrange for which the well known Kolmogorov K41 law, as in [Fri95], is a reasonable approximation.

Synthesis To synthesize isotropic turbulence, frequency-matched curl noise (Section 2.4.1) is used. Similar to [KTJG08] the spectrum is divided into N octaves and each band is synthesized using the curl of band-limited wavelet noise [CD05] with band coefficients determined using the K41 law. In all our demos, we uses three octaves. The total velocity $\mathbf{u}$ of a particle is then given by the large scale flow velocity $\bar{\mathbf{U}}$, interpolated from the low resolution Eulerian solver, and the turbulence velocity:

$$\mathbf{u} = \bar{\mathbf{U}} + 2(\alpha_S k)^{\frac{1}{2}} \sum_{i}^{N} \mathbf{c}_i(\mathbf{q})\, 2^{-\frac{5}{6}i} \tag{4.15}$$

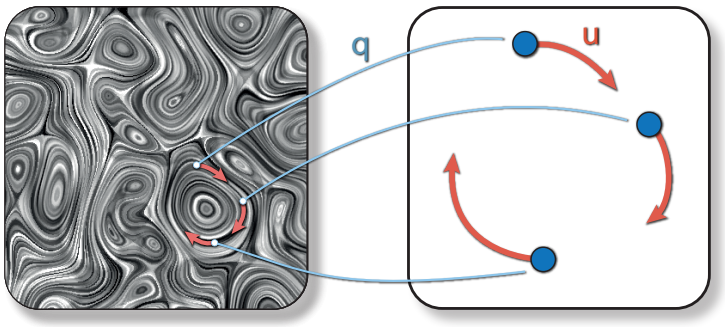


Figure 4.8: Each particle owns a lookup coordinate into the turbulence velocity texture. Both the particle position and the texture coordinate are advected by this velocity, allowing coherent eddies to form.

Here, $\mathbf{c}_i(\mathbf{q}) = \nabla \times \mathbf{N}_i(\mathbf{q})$ are the curl noise textures as described in Section 2.4.1. $\mathbf{q}$ represents a texture coordinate, and k denotes the energy of the largest-scale turbulence band. Instead of storing the texture coordinate on a grid as in Kim et al. [KTJG08], in our case the coordinate is stored directly on the particle system. As we only model forward scattering, we can assume the turbulent detail map is passively advected in the large-scale flow. This means that both particles position and the texture coordinate are updated using the turbulent detail velocity, but only the particle position is advected using the large-scale flow velocity $\bar{\mathbf{U}}$. This allows the formation of coherent turbulent whirls over multiple particles, as shown in Fig. 4.8.

The scaling of the wavelet turbulence is chosen such that the largest synthesized vortices cover 2–4 grid cells, as vortices on these scales are usually dampened out by numerical viscosity of the Eulerian simulation which is better able to represent larger vortices. The energy for the synthetic turbulence is directly given by the $k-\varepsilon$ model with $k = k_{iso} + k_{an}$ for each particle, where k_{iso} denotes the isotropic energy, and k_{an} the anisotropic energy. Assuming $k_{an} = 0$ for now, the energy for the largest band is given by $\alpha\, k_{iso}$. The scaling parameter α encodes the shape of the assumed energy spectrum, and can be used to artificially increase or decrease the strength of the turbulence. This is one of the two main tuning parameters of this model.

Anisotropy

So far we have only considered isotropic turbulence. Some important effects, however, most notably the production of turbulence, are highly anisotropic. Neglecting anisotropy would therefore result in turbulence structures that are not fully connected to the motion of the underlying base flow. Capturing anisotropy requires both a way to synthesize anisotropic noise, as well as an energy transport model capable of providing the anisotropic energy distribution. For the latter, Reynolds stress transport models as mentioned in Section 2.2.2 can be used. These models describe the evolution of tensor quantities, most notably the Reynolds stress tensor. While the complete model is far too complex to be applied in real-time applications, we will use selected elements of this theory to augment our model. Ac-

cording to [Pop00], the most relevant case of anisotropy is the production of elongated vortex structures due to shear effects. This happens, e.g., at boundaries and leads to rotational velocities perpendicular to the shear plane, reducing the dimensionality of the effect from three to two dimensions. Therefore, we consider the case of turbulence consisting of an isotropic component with energy k_{iso} that is handled as described above, and a completely anisotropic two-dimensional component with the energy vector $\mathbf{k_A}$. The direction of $\mathbf{k_A}$ defines the normal of a plane to which the anisotropic turbulence is confined. This is equivalent to a preferred rotation axis, and the magnitude of $\mathbf{k_A}$ defines the energy contained in the anisotropic vortices.

Energy Dynamics While the k–ε equations (4.13),(4.14) still hold for the total energy $k = k_{iso} + k_{an} = k_{iso} + |\mathbf{k_A}|$, we need to determine the evolution equation of the anisotropic component $\mathbf{k_A}$. The mechanisms here are similar to transport of total energy: there is a production, a dissipation and additionally, a redistribution term. Analogous to (4.12), the production vector is given by the turbulent viscosity ν_T and the strain. We use an eigendecomposition to divide the strain tensor into an anisotropic component, represented with two-dimensional turbulence, and an isotropic component. In the following, λ_i denote the eigenvalues and $\mathbf{v_i}$ the eigenvectors of S_{ij}, where λ_1 has the biggest and λ_3 the smallest absolute value. Now consider the production ellipsoid defined by the vectors $\mathbf{p_i} = 2\,\nu_T\,\lambda_i^2\mathbf{v_i}$. The plane of two-dimensional shear stress is spanned by its two longest vectors $\mathbf{p_1}$, $\mathbf{p_2}$, while the plane normal is given by $\mathbf{v_3}$. The isotropic component, on the other hand, is the sphere spanned by the smallest common component of all vectors, that is $|\mathbf{p_3}|$. Therefore, we can define the anisotropic production vector to be

$$\mathbf{P_A} = 2\,\nu_T\,(\lambda_1^2 + \lambda_2^2 - 2\,\lambda_3^2)\,\mathbf{v_3} \quad . \tag{4.16}$$

While in areas of high production, e.g. near obstacle boundaries, anisotropic effects can be observed, the turbulence further away from these regions is largely isotropic. This is due to the fact that transport processes lead to a quick isotropization of the turbulent flow. This is true for spatial turbulent transport as well for transport through the energy cascade. The LRR-IP model mentioned in Section 2.2.2 models this isotropization. If we transfer this model to our turbulence setting, it yields an energy transfer rate of

$$\frac{\mathrm{D}\mathbf{k_A}}{\mathrm{D}t} = (1 - C_A)\mathbf{P_A} - C_R\frac{\varepsilon}{k}\mathbf{k_A}\,. \tag{4.17}$$

from $|\mathbf{k_A}|$ to k_{iso}. The standard model constants are defined as $C_A = 0.6$, $C_R = 1.8$. Here, the dissipation ε is generally assumed to be isotropic, as it occurs on very small scales, while the anisotropic vortices are initiated primarily on the larger scales. This means that it is sufficient to solve the isotropic (4.14) for dissipation.

Synthesis We now extend the synthesis algorithm from Section 4.2.2 for anisotropy by including additional turbulence bands. As the turbulent kinetic energy k is composed of an isotropic component k_{iso} and an anisotropic component $\mathbf{k}_A$, we will compose the synthesis term using an isotropic and an anisotropic part. The isotropic part is equivalent to (4.15) with k_{iso} instead of the total energy k. For the anisotropic component, on the other hand, the

2D curl noise field

$$\mathbf{c}^{2D}(\mathbf{q},\mathbf{k_A}) = \nabla \times N(\mathbf{q})\,\frac{\mathbf{k_A}}{|\mathbf{k_A}|} \tag{4.18}$$

is used. It is aligned to $\mathbf{k}_A$, and thus generates turbulent eddies in the plane normal to the anisotropy vector. For easier precomputation, we effectively use the field $\mathbf{c}^{2D}(\mathbf{q}) = \mathbf{c}^{2D}(\mathbf{q},\mathbf{e}_z)$ and then apply the rotation operator $\mathbf{R}(\mathbf{k}_A) = \mathrm{Rot}(\mathbf{e}_z \leftarrow \mathbf{k}_A)$ during the lookup. The total velocity $\mathbf{u}$ of a particle can therefore be determined by the equation

$$\begin{aligned}\mathbf{u} = \bar{\mathbf{U}} + 2(\alpha k_{iso})^{\frac{1}{2}} \sum_i^N \mathbf{c}_i(\mathbf{q})\, 2^{-\frac{5}{6}i} + \\ 2\,|\alpha\,\mathbf{k_A}|^{\frac{1}{2}} \sum_j^M \mathbf{R}(\mathbf{k_A})\,\mathbf{c}_j^{2D}(\mathbf{q})\, 2^{-\frac{5}{6}j}\end{aligned} \tag{4.19}$$

with $k_{iso} = k - |\mathbf{k_A}|$. As anisotropy decays quickly in the spectral cascade, we have found that it is sufficient to use one band of anisotropic turbulence for the largest scale.

4.2.3 Implementation

For real-time performance, both the Eulerian fluid simulation and the particle based turbulence model run on the GPU. Even if the base simulation can be lightweight and quickly calculated on the CPU, the memory transfer overhead is too big for interactive scenarios.

For the underlying Eulerian solver, we use a typical MAC discretization with second order semi-Lagrangian advection, as described in [Bri08] and [SFK+08]. As PCG, the standard technique for solving the pressure projection on the CPU does not scale well on GPU hardware, the following examples makes use of a GPU multi-grid solver for computing the pressure correction, as described in [CTG10]. For the Lagrangian turbulence model, each particle stores its position and velocity as well as the turbulence parameters $k, \varepsilon, \mathbf{k_A}$ and $\mathbf{q}$. The evolution of these variables is given by integrating (4.13), (4.14), (4.17), and evaluating (4.19), respectively, on the particle system. We use a simple forward Euler integrator for all of these equations. The strain eigen-decomposition required for (4.16) is calculated on each grid cell. As the 3×3 strain tensor is symmetric, eigenvalues can be found efficiently using the analytic formulation [Smi61]. Our model is designed to work without any particle-particle interactions, and only a few linear interpolations of data from simulation grid for velocity and strain are necessary to compute the particle dynamics. This makes it very efficient to compute even in massively parallel settings. In our setup, the smoke is rendered online using half-angle volumetric shadowing by Ikits et al. [IKLH04], enabling the complete framework to run at interactive frame-rates and therefore providing immediate results. Below, we will discuss important details concerning stability and initialization.

Stability The k–ε model, being a coupled system of two PDEs in its original form, has inherent stability problems. Especially k in the denominator of (4.14) causes instabilities for flows with low turbulence. Therefore, the model is usually modified to guarantee stability. A commonly used approach is a low-Reynolds number treatment, as described in [Pop00]. We use a simplified version of this approach to ensure that a minimal turbulent energy is always present in the simulation.

A meaningful range for the turbulent energy k is given by $k = \frac{3}{2}U_0{}^2 I^2$, with the turbulent intensity $I \in [0 \dots 1]$. Here, U_0 is the characteristic velocity scale, which can be determined from the simulation parameters, e.g., the maximal speed of the car in Fig. 4.5. As suggested in the field of aerodynamics research [SR07], we use a value of $I_{min} = 10^{-3}$ as a minimal turbulent intensity, while, naturally, the maximal intensity is given by $I_{max} = 1$. By restricting the simulation to this meaningful range of values, the system quickly recovers from overshoots and is stable for arbitrary time steps.

Similarly, we can define a corresponding range for the values of ε. We obtain a minimum dissipation by specifying a minimal turbulent viscosity equal to the molecular viscosity of air ν_{air}, which represents a natural lower bound for the viscosity of smoke simulations. Using the definition of turbulent viscosity (2.9) for the k–ε model, we obtain $\varepsilon_{min} = C_\mu \frac{k_{min}{}^2}{\nu_{air}}$. The maximal dissipation, on the other hand, can be derived on the basis of a minimal turbulent length scale L_{min}, and is given by $\varepsilon_{max} = C_\mu{}^{\frac{3}{4}} k_{max}{}^{\frac{3}{2}} \frac{1}{L_{min}}$. We use a minimal length scale of $\frac{1}{10}$ of a grid cell in our simulations.

Note that these ranges for turbulent energy and dissipation are useful when allowing users to interact with the simulation. They can, e.g., provide artists with intuitive parameter ranges for setting up turbulence sources in a scene.

Initial state We seed the particles at the smoke inflow of the scene. As this will usually not coincide with the fluid inflow region, we need to specify sensible initial values for the turbulence parameters of these particles. In cases where the inlet is in a low-turbulence area, we can use the lower boundaries k_0 and ε_0 as initial values. If, on the other hand, the smoke should be generated in a turbulent region, we need to specify initial energy levels, as we have no information about the history of the particles. This can be achieved with different approaches. We estimate typical turbulent intensities for k and ε similar to the estimation of maximal bounds described in the previous paragraph, and use these values for initializing the particles. Here, the minimal length scale L_{inlet} is another important parameter of our model, and can be used to tune the amount of turbulence injected into the scene. Another possibility is to initialize particles with the lower bounds k_0 and ε_0, and then perform a small number of iterations of the turbulence model on the newly seeded particles.

Texture Advection Naturally, the structure of the turbulence should deform as given by the motion of the flow. However, using a naive approach, e.g., updating the local texture coordinate $\mathbf{q}$ of each particle using only the large scale motion with $\frac{\mathrm{D}\mathbf{q}}{\mathrm{D}t} = \mathbf{u} - \bar{\mathbf{U}}$ quickly destroys coherence of the turbulent structures. By compression and mixing in flow, adjacent particles will eventually own strongly divergent texture coordinates. This destroys coherent structures as in Fig. 4.8 and will inevitably lead to uniform noise instead of recognizable swirling motions. This loss of coherence is closely related to the problem of texture field deformation in methods such as Wavelet Turbulence (Section 4.1.4). We however do not want to rely on local resetting, as by construction, our aim is to update each particle without having to know about its neighbors. It is therefore undesirable to perform any kind of spatial interpolation on the particles.

There are two viable approaches to combat this problem. The first are *guiding particles*, as presented in the original paper [PTC+10]. They to preserve the local coherence of the

1: *// Grid-based Fluid solver*
2: Semi-Lagrangian advection of $\bar{\mathbf{U}}$
3: Pressure projection
4: Calculate strain field S_{ij}
5:
6: Seed and initialize new particles
7:
8: **for each** particle **do**
9: Sample U, S_{ij} at particle position $\mathbf{x}$
10:
11: *// Energy dynamics*
12: Compute turbulent viscosity: $\nu_T \leftarrow C_\mu \frac{k^2}{\varepsilon}$
13: Compute production: $\mathbf{P} \leftarrow$(4.16)
14: Integrate $k \leftarrow k + \Delta t\,(|\mathbf{P}| - \varepsilon)$
15: Integrate $\varepsilon \leftarrow \varepsilon + \Delta t \frac{\varepsilon}{k}(C_1 P - C_2 \varepsilon)$
16: Energy transfer: $\mathbf{k_A} \leftarrow \mathbf{k_A} + \Delta t\,(1 - C_2)\mathbf{P} - \Delta t\, C_R \frac{\varepsilon}{k}\mathbf{k_A}$
17: Stabilize k, ε using $k_{min,max}$ and $\varepsilon_{min,max}$
18:
19: *// Motion equations*
20: Synthesize velocity: $\mathbf{u} \leftarrow$ (4.19)
21: Integrate $\mathbf{x} \leftarrow \mathbf{x} + \Delta t\,\mathbf{u}$
22: Integrate $\mathbf{q} \leftarrow \mathbf{q} + \Delta t\,(\mathbf{q_G} + \mathbf{x} - \mathbf{x_G})$
23: **end for**
24: Advect guiding particles in flow field $\bar{\mathbf{U}}$
25:
26: Render simulation data

Figure 4.9: Pseudo-code for the simulation loop.

turbulence. Guiding particles are seeded together with the actual smoke particles, and assigned to a small group of smoke particles based on local neighborhood. On seeding, each guiding particle is assigned a fixed texture coordinate $\mathbf{q_G}$ based on its world coordinate, which acts as a local frame of reference for the texture coordinates of the attached smoke particles (Fig. 4.10).

As the guiding particles represent the motion of the turbulence textures, they are advected using only the large scale flow from the underlying simulation. The local texture coordinate of each smoke particle can now be computed using the local position $\mathbf{x}$ with respect to the assigned guiding particle as $\mathbf{q} = \mathbf{x} - \mathbf{x_G} + \mathbf{q_G}$. This approach allows us to efficiently preserve locality while adhering the turbulence motion to the large scale flow. While coherence and incompressibility are exactly preserved within the particle cloud of a guiding particle, coherence loss and small-scale deviations from incompressibility may appear between these clouds. Therefore, guiding particles should be seeded such that the associated clouds are compact, sized above turbulence length scale, and cover all flow paths. In the following example scenes, we seed between 1 and 10 guiding particles per timestep,

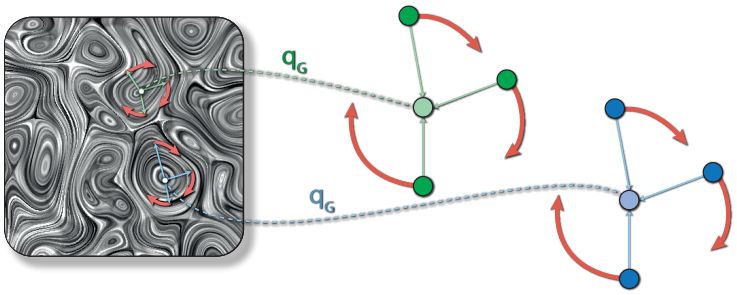


Figure 4.10: Two groups of particles (blue, green) with an associated guiding particle are shown. The texture coordinate lookup of the individual particles is performed by taking the geometric distance to the guiding particle, and the associated guiding texture coordinate $\mathbf{q}_G$, which is the same for the cluster. This way, texture coordinates will stay coherent within the cluster.

randomly distributed across the seeding area.

For complex flows, it can be hard to find a good compromise on guiding particle density. In our experience, the approach of alternating global coordinate resetting as described in Section 2.4.2 using two sets of texture coordinates is often the better choice; While it introduces some artificial diffusion, it provides coherence even under strong deformations and is easier to tune. If both methods prove unsufficient, it might also be worthwhile to investigate more sophisticated models for texture advection, e.g., [YNBH09].

The complete simulation loop is specified in the pseudo-code in Fig. 4.9.

4.2.4 Results and Discussion

In the following, we will discuss several simulations to highlight the features of this model and differences to previous work.

Comparison with reference simulation In order to evaluate realism, we simulate the flow in the wake of a car (Fig. 4.11). The simulation uses 1M particles, and a base solver resolution of $32 \times 8 \times 32$. We compare our model to a $256 \times 64 \times 256$ high-resolution reference solver. While the exact form of the turbulence is different between our method and the reference solver, we observe that both show a similar level of small-scale detail.

Energy model The energy model is tested by simulating a flow over a ramp shown in Fig. 4.13. This setup uses a resolution of the base solver of $64 \times 16 \times 16$ grid cells. When using low grid resolutions such as this, flow instabilities induced by obstacles are dampened out, and no turbulence is induced. This effect can be seen in the top image of Fig. 4.13. In this example, turbulence should develop to the left of the ramp as the flow travels from right to left. Our method tracks causality in the production of turbulence, resulting in a correct swirling motion perpendicular to the edge of the step, purely behind the sharp edge. Turbulence synthesis methods such as Wavelet Turbulence that amplify or derive turbulent energy directly from the computed velocity field do not track the causality in the production of

Figure 4.11: The wake behind a car is simulated with 1M particles. Our method (top) and the reference high-resolution solver (bottom) show similar small-scale details.

turbulence. In this case, Wavelet turbulence incorrectly produces turbulence in the laminar region right of the edge.

In a more complex example shown in Fig. 4.7, we simulate a train accelerating and braking. Here, the source of turbulence is not induced by obstacles, as in the ramp example, but is due to the pulsed emission of smoke from the chimney. This is also inherently handled by the production term of our energy model. Also, correct adaptation of turbulence intensity to the train's velocity can be observed.

Anisotropy The effect of anisotropic turbulence is demonstrated in a simulation of a strongly turbulent flow past a cylinder. We seed a thin horizontal sheet of smoke to the right, visualizing only a slice of the 3D problem. The side-view of the simulation, with anisotropy handling disabled (top) and enabled (bottom) is shown in Fig. 4.12. If anisotropy is not handled, isotropic turbulence is injected immediately downstream of the obstacle. This leads to strong disturbances normal to the plane of motion, as can be seen in the top image of Fig. 4.12. Our model predicts a zone of high anisotropy behind the cylinder. Here, the turbulence is expected to be confined within the smoke sheet, therefore integrating with the large scale Karman vortices, before becoming more and more isotropic, towards the left side of the lower image.

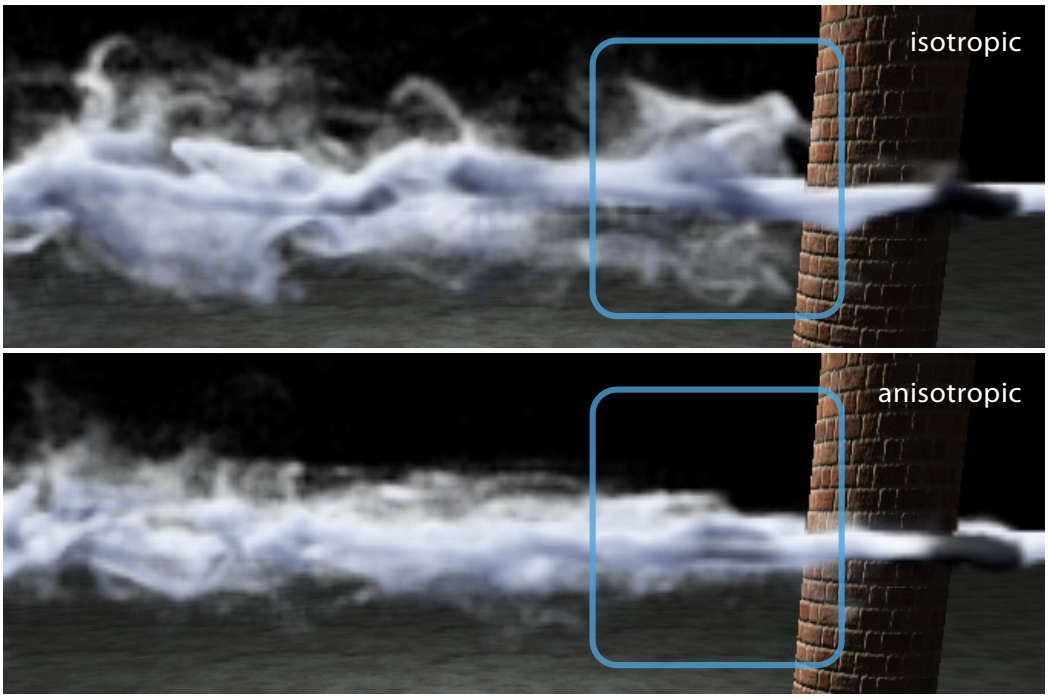


Figure 4.12: In this example, a thin sheet of smoke flows around a cylinder. Here, the side view is depicted. Using only the isotropic turbulence model (top), the induced turbulence disturbs the flow, as can be seen by the unrealistic spikes left of the cylinder. Using our anisotropy extensions (bottom), the turbulence integrates into the overall flow, and a smooth transition to full isotropic turbulence can be observed.

Scalability To demonstrate the scalability of our model, we simulate a smoke wake behind a car with varying particle numbers, while keeping the grid resolution fixed at $32 \times 8 \times 32$. As can be seen in Fig. 4.5, the large scale flow remains consistent in all cases, while the amount detail is controlled by the number of particles. As it is sufficient to use a very low grid resolution for the Eulerian solver in all examples, the performance scales approximately linearly in the number of particles. With one million particles, our model achieves 15 frame per second on average (including rendering). Increasing the number of particles to four millions, we still achieve 4.7 frames per second. The exact numbers can be found in Table 4.1. This means this model is able to compute accurate turbulence dynamics efficiently. GPU-based methods relying on grids are strongly limited in detail due to the available memory, the Eulerian solver of our implementation, e.g., is limited to a 128^3 resolution using the same hardware. Using the particle based approach we are, on the other hand, able to achieve very detailed motion in an efficient manner.

Our method also opens up the possibility to compute and synthesize turbulence outside the grid-based solver. If no underlying grid is present, zero turbulence production and the last encountered large-scale velocity are taken as an input for the calculation. This allows a smoke volume to leave the domain of the Eulerian simulation, while still exhibiting turbulent motion, as shown in Fig. 4.14. This is very useful for interactive applications

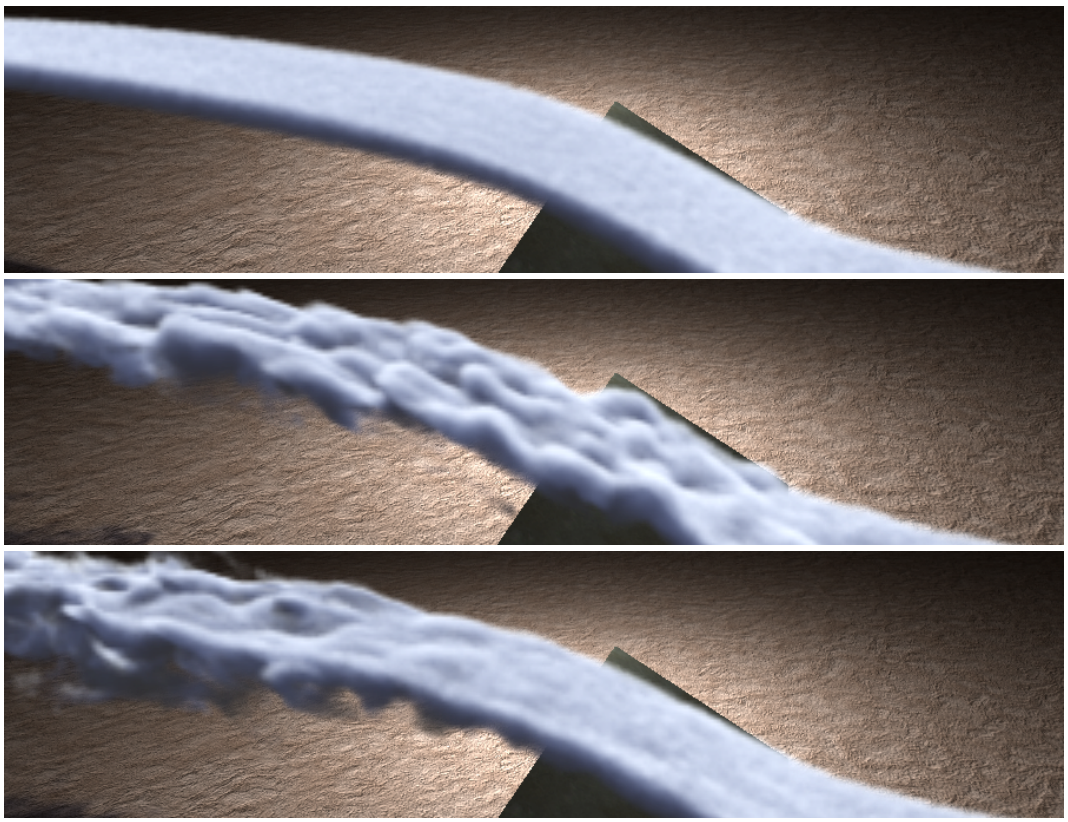


Figure 4.13: A flow over a ramp is simulated. The low-resolution solver (top) does not represent the flow instability after the edge of the ramp. Therefore methods like Wavelet Turbulence (middle) that depend on the solver for energy calculations also fail to catch the correct turbulence seeding region. Our model (bottom) is able to predict turbulence production due to a full energy transport model.

where the spatial limits of the domain should be hidden from the user.

For all of our examples, we vary only the α and L_{inflow} parameters. Recall that α controls the overall amount of turbulence and L_{inflow} controls turbulence at an inlet. Varying only these parameters allows for artistic control while retaining visual realism.

4.2.5 Conclusions

In this chapter, a scalable algorithm for simulating anisotropic turbulence was introduced. By separating the system into a grid-based solver and a decoupled particle system without particle-particle interactions, our method is highly efficient on parallel systems. The algorithm is driven by an anisotropic energy transport mechanism, and handles both free stream turbulence production and turbulence induced by walls. Turbulence is synthesized directly on the rendered particles, which allows the simulation to handle the full detail that will later on be displayed, while not wasting any processor cycles for regions that are not visible. This way, we achieve frame rates of more than 15 frames per second even for detailed simulations with millions of particles.

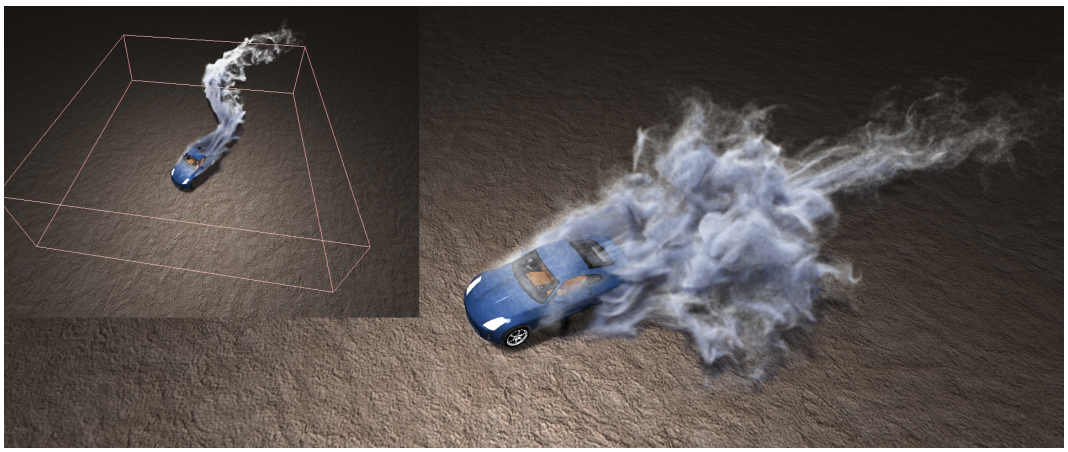


Figure 4.14: The smoke in this turbulent wake is represented using a particle system. As we synthesize turbulence directly on the particle system, the particles may leave the simulation domain, shown as a red box in this image.

On the other hand, the approach is restricted to single phase fluid simulations. It does also not perform well for large, non-turbulent smoke volumes, which can have unnecessarily large numbers of particles inside the volume that hardly move.

An interesting addition to this model could be the adaptation in a LOD sence for large interactive scenes, as the modularity of our approach makes it highly suitable to combine different simulation approaches. This will allow a smooth transition from a simple static flow field, to a Eulerian fluid simulation, while finally adding detail with the anisotropic Lagrangian turbulence model.

A limitation is that this algorithm can exhibit artifacts when the underlying simulation is not able to resolve all features of a flow, e.g., in the presence of very thin objects.

Finally, this approach shares the limitations of all statistical turbulence synthesis methods, in that turbulence transition is not well-represented. The breakdown of coherent structures to turbulence is not easily modeled in a statistical manner. As our extension for anisotropy guarantees that the overall shape of the turbulence distribution behaves accordingly, it is able to produce convincing results for fast-developing turbulence. The statistical approach will fail, however, for slow turbulence breakdown, which is visible e.g. in the interface of dense buoyant smoke plumes, or in instable flows.

The methods developed in the the following chapters will address this inherent issue. By directly modeling the turbulence transition process using vortex methods, they allow to represent a wide range of turbulent effects that have not been accessible to turbulence methods.

Setup	Grid res.	#part	α	L_{in}	Base [ms]	Part. [ms]	Total [fps]
Car (Fig. 4.5)	$32 \times 8 \times 32$	250k	2.5	0.04	20	7.6	34
Car	$32 \times 8 \times 32$	1M	2.5	0.04	19	27	15
Car	$32 \times 8 \times 32$	4M	2.5	0.04	20	92	4.9
Car (no turb.)	$32 \times 8 \times 32$	1M	–	–	19	6.4	20
Ramp (Fig. 4.13)	$64 \times 16 \times 16$	1M	2.6	0.08	19	23	17
Aniso. (Fig. 4.12)	$64 \times 16 \times 16$	1M	15.0	0.1	19	27	15
Iso. (Fig. 4.12)	$64 \times 16 \times 16$	1M	15.0	0.1	14	27	16
Smoke gun	$48 \times 48 \times 48$	1M	4.2	0.02	25	18	18
Train (Fig. 4.7)	$64 \times 32 \times 16$	6M	3.0	0.05	44	161	3.7

Table 4.1: Performance numbers for our simulation runs. Timings are given per frame. *Base* refers to the grid-based solver, while *Part.* represents turbulence computation, synthesis and particle system update. The total framerate includes both simulation and online rendering. All simulations were run on a NVidia GTX 480 graphics card on a workstation with an Intel Core i7 CPU and 8GB of RAM.

4.3 Obstacle-Induced Turbulence

Many interesting forms of turbulent flow originate from the complex interaction of flows with obstacles. At the wall of these obstacles, a very thin boundary layer forms, which may separate from the wall, become instable and form free turbulence. As these processes occur on a lengthscale below the resolution of most simulations, the turbulence generation process is often misrepresented or simply omitted. Even turbulence reference scenarios from CFD, such as driven cavities or passive grids are only represented correctly using very expensive high-resolution simulations.

In this chapter, we will introduce a method to model this process and predict turbulence generation by flow obstacles, as presented in [PTSG09]. Instead of synthesizing turbulence using a frequency-matched curl noise texture as in Section 4.2, we will directly represent the turbulence eddies using vortex particles, which enables coherent anisotropic structures in e.g. turbulence transition. Together with prediction, this will allow us to represent a wider range of complex turbulence effects, which can not be achieved using the general-purpose turbulence model presented in the previous chapter. As we are able to precompute the formation process for a given obstacle geometry, we can also predict turbulence generation from obstacles thinner than grid resolution (Fig. 4.15).

While turbulence detail enhancement using vortex particles has been previously studied in Graphics [SRF05], these methods deal with the preservation of vortices manually injected in the flow. We use an enhanced version of the vortex particle approach and combine it with turbulence methods to guarantee that particles are accurately seeded and the energy dynamic adheres to what is predicted by CFD turbulence theory. Moving the computations for the generation of turbulence into a preprocessing step allows us to quickly set up new simulations around a given object.

Figure 4.15: This algorithm allows us to precompute detailed boundary layer data and efficiently reuse it for new simulations. We are able to generate turbulent vortices taking into account the relative velocity of an obstacle in the flow. Here, we apply the algorithm to a very thin object that is barely represented on the simulation grid.

4.3.1 Overview

The algorithm presented consists of a precomputation step for the scene geometry, and a simulation step.

The precomputation step captures the characteristics of the boundary layer around the object, and stores it for different sets of flow directions. This allows us to purely resolve the geometry of the object with the precomputation, instead of having to fully resolve the actual flow velocity in the often very thin boundary layer. For this precomputation, we assume that an object can be characterized by a relative translational and rotational velocity, allowing for simulations of rigid body motion or static flows of arbitrary direction.

The main simulation method consists of a standard, grid-based fluid solver, e.g. according to Stam [Sta99], augmented with a turbulence representation. The precomputed boundary layer data is used to efficiently calculate where vortices are created around the object in a separate simulation. We compute the evolution of boundary layer around the object, and estimate regions where this field becomes unstable to form actual turbulent vortices. The turbulent vortices are represented using an improved variant of vortex particles [SRF05], which induce rotation in the flow around the particle position. While the vortex particles are created based on boundary layer vorticity, their dynamics is based on the vorticity equation. In an additional step, we re-mesh the particles and adjust the particle

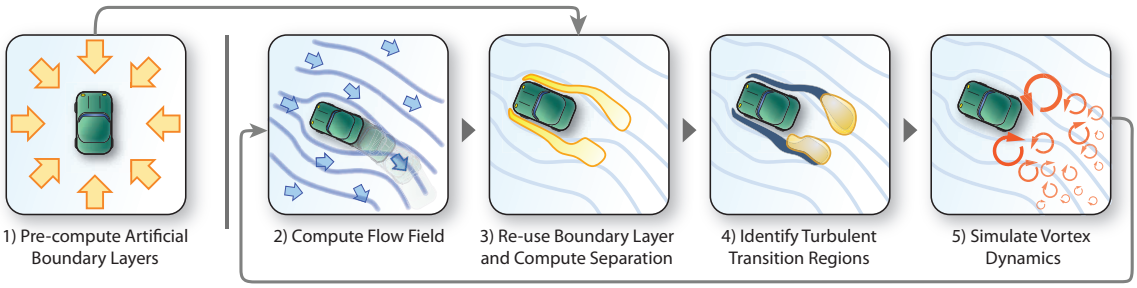


Figure 4.16: An overview of different steps of the algorithm. After precomputing the artificial boundary layer (1), we run or not a new simulation (2) and apply the confined vorticity from the precomputation (3). Regions transitioning into turbulence are identified with an approximation of the Reynolds stress (4). This results in the creation of vortex particles. Their dynamics are computed in an additional step (5).

kernel to ensure a correct turbulent energy distribution. The resulting vorticity is finally reconstructed onto a grid with higher resolution than the base simulation.

The key point of this chapter is the turbulence estimation and vortex particle seeding mechanism. In Section 4.3.3, we will develop a theory for this. In Section 4.3.4, the dynamics of the vortex particles are described, and the coupling of the vortex particles to the flow field is explained. The actual simulation loop and implementation details are discussed in Section 4.3.5. The simulation loop is also visualized in Fig. 4.16. Finally, we will evaluate the method and present results in Section 4.3.6.

4.3.2 Vorticity Formulation

Vorticity is defined as the curl of the velocity field $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ and is a description of flow rotation. This representation is advantageous especially for describing turbulent flows, which consist of small rotational whirls, as these structures have a more compact support in vorticity than in velocity formulation. By applying the curl operator to the NS equations, we obtain the *vorticity equation*

$$\frac{\mathrm{D}\boldsymbol{\omega}}{\mathrm{d}t} = \boldsymbol{\omega} \cdot \nabla \mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} + \frac{1}{\rho} \nabla \rho \times \left(\mathbf{g} + \frac{1}{\rho} \nabla p \right). \tag{4.20}$$

The substantial derivative on the left-hand side includes vorticity advection, while the right-hand side consists of the vortex stretching, diffusion and baroclinity terms. Vortex stretching describes the deformation of the vorticity field under the influence of the velocity field, while baroclinity models its behavior across density gradients and gravity. One implication of this baroclinity term is buoyant movement, which in the vorticity formulation is represented by a sheet of vorticity at the density gradient or interface, yielding e.g. the typical vortex rings for rising smoke.

The continuity equation (2.2) on the other hand is implicitly satisfied, as rotational fields are divergence-free. We also note that the pressure term vanishes for flows with constant

densities. However, advection and vortex stretching require a velocity field, which has to be obtained by integrating the vorticity field.

As the Helmholtz theorem shows, we can decompose a given velocity field $\mathbf{u}$ into a curl-free component $\mathbf{u}_\Phi$ and a divergence-free component $\mathbf{u}_\Psi$. The divergence-free component can be related to a vector potential Ψ and the vorticity by

$$\mathbf{u}_\Psi = \nabla \times \Psi \,, \qquad \boldsymbol{\omega} = \nabla \times \mathbf{u}_\Psi \quad .$$

Applying the continuity equation for incompressible fluids, we find that $\mathbf{u}_\Phi$ has to be constant. Therefore the velocity field is completely described by vorticity or the vector potential Ψ. This is strictly only true for the free-space case, however. With limited domains, as encountered in all grid-based simulations, additional terms are needed to describe the behavior at the domain boundaries, if velocity does not vanish there. Finally, to integrate the velocity field induced by vorticity we can use the free-space solution to the rotation operator, the Biot-Savart law

$$\mathbf{u}_\Psi(\mathbf{x}) = \frac{1}{4\pi} \int \boldsymbol{\omega}(\mathbf{x}') \times \frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3} \, d\mathbf{x}' \quad . \tag{4.21}$$

Vortons The most popular and well-researched primitive to represent vorticity is the point element *Vorton* [Hal79]. Each vorton i owns a position $\mathbf{x}_i$ and its associated vorticity $\boldsymbol{\omega}_i$. The total vorticity field of the system is given by

$$\boldsymbol{\omega}(x) = \sum_i \boldsymbol{\omega}_i \delta(\mathbf{x}-\mathbf{x}_i) \tag{4.22}$$

using Dirac's delta function δ. In many methods, an additional particle radius or kernel is used. Vortons are the most general primitive and are especially suitable for highly turbulent flows. In these flows, coherent structures break down to small isolated vortices, and the role of connectivity is diminished.

The motion equation for the vortons is given by (4.20). While advection is handled implicitly, the right-hand side terms are not easily expressed in terms of a particle system. The vortex stretching term can be evaluated by explicitly calculating the velocity gradient tensor at the particle position. However, this has been shown to produce divergent flow fields [CK99]. One way to regularize this problem is to use an intermediary grid [MG96]. The diffusion term can be implemented by core-spreading, that is increasing the particles radius [Leo80] or particle strength exchange methods [DMG89]. Full vorton models require overlapping particles for convergence, therefore re-meshing is needed to ensure full coverage. While local re-meshing methods do exist [SvD96], most vorton methods use a form of global re-meshing [CK99].

The most direct approach to obtain a velocity field from a vorton system is the discretization of (4.21)

$$\mathbf{u}_\Psi(\mathbf{x}) = \frac{1}{4\pi} \sum_i \boldsymbol{\omega}_i \times \frac{\mathbf{x}_i-\mathbf{x}}{|\mathbf{x}_i-\mathbf{x}|^3} \quad . \tag{4.23}$$

This equation is however singular for points close to vortons. Chorin et al. [CB73] therefore introduced a regularization mechanism

$$\mathbf{u}_{reg}(\mathbf{x}) = \frac{1}{4\pi} \sum_i \boldsymbol{\omega}_i \times \frac{\mathbf{x}_i-\mathbf{x}}{(|\mathbf{x}_i-\mathbf{x}|^2+\alpha_R^2)^{\frac{3}{2}}} \quad . \tag{4.24}$$

The regularization parameter α_R effectively controls the minimal size of the generated vortices. As an alternative to direct integration, the particles can be projected on an auxiliary grid using a smoothing kernel. On the grid, the velocity can be obtained by solving the Poisson equation for the vector potential as in the Eulerian case. This hybrid method is called *Vortex-in-Cell* (VIC) and can be used for all vortex primitives [CP03]. The grid and particle projection kernel act as an implicit regularization, so no additional terms are needed.

In Graphics, Vortons are called *Vortex Particles*, and mostly used in a sparse setting. Therefore, less strict re-meshing is used. Also, the diffusion term is commonly ignored, for the same reasons as in the velocity form of the NS equations. The original approach for vortex particles [SRF05] does not use velocity integration using the Biot-Savart law, but a confinement force that acts on the underlying simulation, in a similar manner as vorticity confinement [FSJ01]

$$\mathbf{F}(\mathbf{x}) = \varepsilon \sum_i \frac{\mathbf{x}_i - \mathbf{x}}{|\mathbf{x}_i - \mathbf{x}|} \times \boldsymbol{\omega}_i \, \delta(\mathbf{x}_i - \mathbf{x}) \tag{4.25}$$

where ε is the confinement strength. This treatment ensures only vorticity not already present in the underlying simulation is added, but suffers from instabilities if ε is chosen inappropriately. We therefore chose to use a modified version of this vortex particle concept, which will be introduced in Section 4.3.4.

4.3.3 Wall-Induced Turbulence

While turbulence in flows is generated by various processes, a very common and visually important one is turbulence generation at the flow boundaries. Therefore, our algorithms explicitly model this important process. In this section, our turbulence estimation method is introduced. It bases on turbulence modeling and wall flow theory, which is introduced in Section 2.2. A more detailed account can be found in the book by Pope [Pop00].

Generation of turbulence

In wall-bounded flows, wall friction enforces a tangential flow velocity of zero at the wall. This leads to the formation of a thin layer with reduced flow speed, called the boundary layer. Fig. 4.17 shows a velocity profile in the boundary layer. It has been shown that this profile is equivalent for all wall-bound flows when using normalized units. This *universal law of the wall* was stated by van Driest in [Dri56].

The gradient of tangential flow velocity in the boundary layer leads to the creation of a thin sheet of vorticity $\boldsymbol{\omega} = \nabla \times \mathbf{u}$. For planar walls, this vorticity remains mostly confined to the boundary layer, and we will thus refer to it as *confined vorticity*. At regions of high flow instability however, vorticity may be ejected from the boundary layer and enter the flow as turbulence. This happens e.g. at sharp edges, where the boundary layer is separated from the wall, and likely to become unstable, or when other turbulent structures disturb the boundary layer. This process of turbulence formation is referred to as roll-up, and is the predominant mechanism of wall-induced turbulence generation [JO93]. There is no theory quantitatively describing the boundary layer roll-up process. We will therefore model this process in a statistical sense, as explained below.

Turbulence modeling As in the previous chapter, we base our approach on CFD turbulence modeling techniques. However, we will only model the region close to the wall (the boundary layer) using RANS-type turbulence models – the free-space turbulence is then handled using a vorticity model. First, we will describe how we model the boundary layer.

Boundary layer modeling In order to accurately model wall-induced turbulence formation, we need to track the confined vorticity, simulate the boundary layer separation and finally identify the transition points to turbulence.

As the boundary layer attached to an obstacle is very thin (smaller than simulation grid resolution in most cases), it is difficult to directly measure the confined vorticity. Instead, we leverage the universal law of the wall, and note that the confined vorticity only depends on the velocity scale and materials constants. For each point in the wall-attached boundary layer we therefore determine the confined vorticity as

$$\boldsymbol{\omega}_{ABL} = \beta(\mathbf{U}_s \times \mathbf{n}) \; . \tag{4.26}$$

The velocity scale $\mathbf{U}_s$ is the tangential component of the averaged flow velocity just outside the boundary layer. The constant β accounts for the two material constants, skin friction coefficient and the fluid viscosity. In our model, β is a user-defined parameter. We call the resulting field $\boldsymbol{\omega}_{ABL}$ the *artificial boundary layer*.

On the other hand, boundary layer separation is an advective transport process. If the wall-attached part of the artificial boundary layer is known, then the separation plume can be derived by advecting this field with the flow field during the simulation run.

The last missing part is to identify regions where the separated boundary layer becomes unstable, and the confined vorticity $\boldsymbol{\omega}_{ABL}$ transitions to free turbulence. The anisotropic part of the Reynolds tensor a_{ij}, which is responsible for the production of turbulence, is a good indicator for such transition regions. We therefore define a transition probability density p_T, which is used to seed turbulence,

$$p_T = c_P \Delta t \, \frac{\|a_{ij}\|}{|\mathbf{U_0}|^2} \tag{4.27}$$

such that regions with high Reynolds stresses are likely transition regions. Here, $\|\cdot\|$ denotes the Euclidean matrix norm. Reynolds stresses are normalized to a uniform scale by the inflow velocity $\mathbf{U}_0$, and c_P is a parameter to control the seeding granularity. If using varying time-steps, p_T has to be multiplied by Δt to ensure consistent behavior. In the following, we will describe how to compute the Reynolds stress tensor based on stresses in the averaged flow field.

Reynolds models The anisotropic component a_{ij} of the Reynolds stress tensor R_{ij} can be expressed using the turbulent viscosity hypothesis

$$a_{ij} = -2\nu_T S_{ij} \; , \tag{4.28}$$

where ν_T is the turbulent viscosity and S_{ij} denotes the strain tensor. The turbulent viscosity can be expressed in terms of a *mixing length* l_m. We chose the model of Baldwin [BL78] for modeling the turbulent viscosity which states

$$\nu_T \approx {l_m}^2 \|\Omega_{ij}\| \; . \tag{4.29}$$

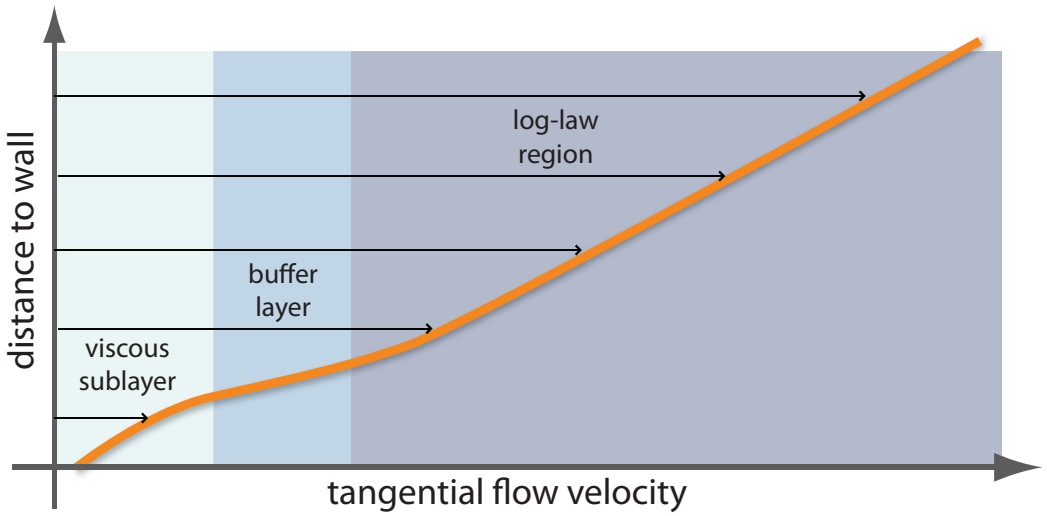


Figure 4.17: The mean velocity profile near a wall (in normalized units) has the form shown above. This has been confirmed in numerous experiments, and was formulated as a universal law by van Driest.

with the rotation tensor Ω_{ij}. While the mixing length is not known for the general case, we only need to model the near-wall region, where l_m is known to be linear in wall distance. We could also have used a complete model such as k–ε – this would however mean modeling the whole domain and tuning inflow parameters, which we can avoid here. Using these standard methods, it is possible to predict the generation of turbulence using only the non-turbulent mean flow velocities.

However, the presented Reynolds stress model requires a high grid resolution around the boundaries to capture the thin boundary layer accurately. In a typical fluid simulation in graphics, the boundary layer thickness is often smaller than a grid cell. Consequently, the discrete S and Ω operators will fail to capture the desired effect, or even cause instabilities due to highly discontinuous gradients, as also mentioned by, e.g. Narain [NSCL08].

We therefore propose two changes to this model. First, we know that in regimes close to a wall, the norm of the rotation tensor equals the norm of the confined boundary layer vorticity, $\|\Omega_{ij}\| = |\boldsymbol{\omega}_{ABL}|$. Also, we assume that $\|S_{ij}\| \approx \|\Omega_{ij}\|$. This is a good approximation if the velocity gradient is dominated by the component normal to the wall [Pop00], which, except for sharp corners, is usually the case in the near-wall region. With these assumptions, we can rewrite the Reynolds stress without the problematic discrete stress and rotation tensors as

$$\|a_{ij}\| \approx 2 {l_m}^2 |\boldsymbol{\omega}_{ABL}|^2. \tag{4.30}$$

Combined with (4.27) this leads to the final equation for the transition probability.

$$p_T = 2 c_P \Delta t \, {l_m}^2 \frac{|\boldsymbol{\omega}_{ABL}|^2}{|\mathbf{U}_0|^2} \, . \tag{4.31}$$

The seeding process for vortex particles, based on p_T, is explained in Section 4.3.4.

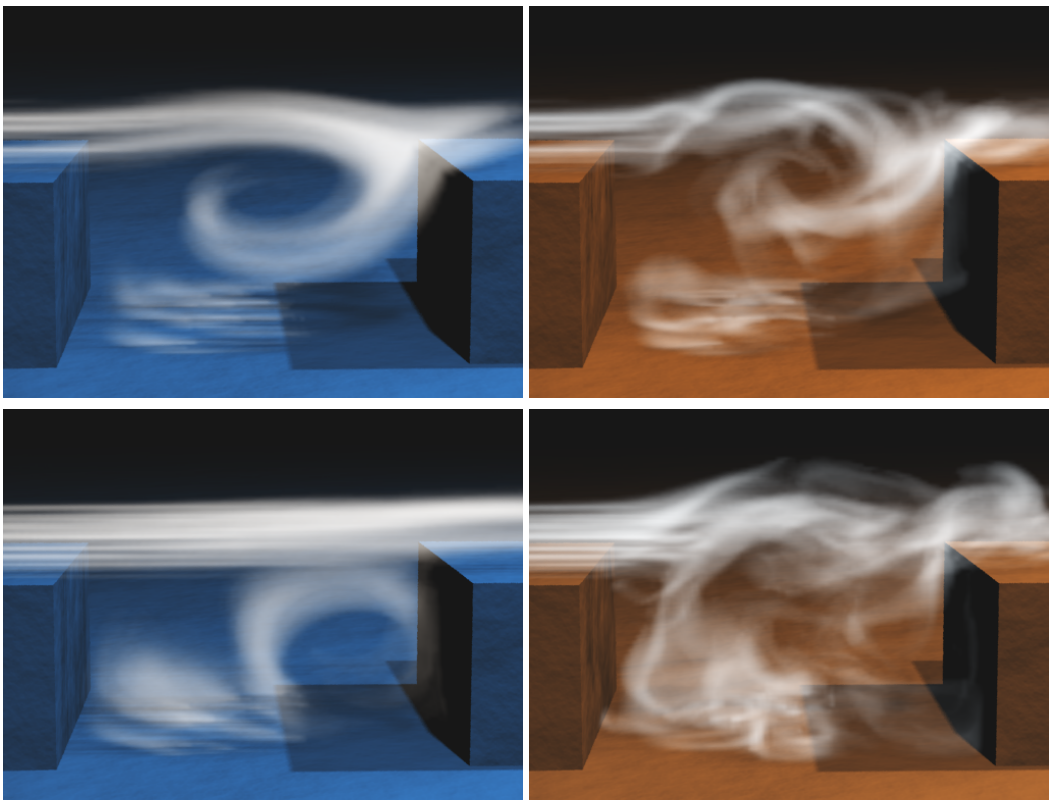


Figure 4.18: The top picture show a basic simulation of a fluid flowing left to right over a cavity. This flow produces a big vortex in the cavity, but is unable to capture any generation of turbulence from the walls. With our method (pictures on the right) we are able to identify the confined vorticity shedding off the two edges of the cavity, and introduce corresponding vortex particles to represent the turbulent structures forming in the flow.

Precomputing the Artificial Boundary Layer

The artificial boundary layer together with (4.31) can be used to seed turbulence, in the form of vortex particles, in the appropriate places of the flow. However, the expression for the wall-attached $\boldsymbol{\omega}_{ABL}$ depends on the averaged flow field $\mathbf{U}$, which is not accessible during the simulation. It is not possible to use the instantaneous flow field of the simulation, as the emerging turbulence would lead to feedback loops. However, we can precompute $\boldsymbol{\omega}_{ABL}$ for quasi-static scenes or scenes with rigidly moving objects. This has the additional advantage that we can choose simulation resolution and precomputation resolution independently, allowing us to precompute fine boundary geometries, while running the simulation on a coarse grid.

Precomputation is done by running a standard fluid solver, and time-averaging the flow field. At all obstacle boundary voxels, (4.26) is evaluated, and $\boldsymbol{\omega}_{ABL}$ is stored in a suitable data structure (see pseudo-code Fig. 4.19). More details on the implementation of the precomputation step, and how the precomputed data is used in the simulation will be given in Section 4.3.5. In the next section, we will explain how to compute the dynamics of our

1: Perform standard grid-based simulation
2: Obtain time-averaged flow field $\mathbf{U}$
3: **for each** voxel $\mathbf{x}$ on the obstacle boundary **do**
4: *// Get voxel outside the boundary layer*
5: $\mathbf{x}_e \leftarrow \mathbf{x} + l\,\mathbf{n}$
6: $\boldsymbol{\omega}_{PRE} \leftarrow \beta\,(\mathbf{U}(\mathbf{x}_e) \times \mathbf{n})$
7: Store $(\mathbf{x}, \boldsymbol{\omega}_{PRE})$ in a point set
8: **end for**

Figure 4.19: Pseudo-code for precomputing the Artificial Boundary Layer. $\mathbf{n}$ denotes the surface normal and l is the boundary layer thickness. l is chosen to be the distance from the wall at which the velocity gradient approaches zero, usually 1-2 grid cells.

turbulence representation.

4.3.4 Turbulence Synthesis

We chose to synthesize turbulence using vortex particles. In contrast to curl noise-texture based turbulence methods, vortex method directly represent the turbulent structures, and therefore automatically preserve coherence. Transition processes can directly be modeled, as they allow more degrees of freedom in anisotropy. Sparse particles allow us to focus on sampling the regions where turbulence is actually generated. Narain et al. [NSCL08] use particles with curl noise textures as a turbulence representation. However, the blending of noise textures creates diffusion, and this approach only supports isotropic turbulence. As we want to model highly anisotropic generation processes, and extend our model into the model-dependent range, where no uniform direction and energy distribution can be assumed, we use an enhanced variant of the vortex particle method by Selle [SRF05] instead. In contrast to the original paper, we also model energy transfer and make use of an improved synthesis step.

Vortex particle dynamics

Turbulence dynamics can be seen from two points of view: The vorticity differential equation describes the direct evolution of the vorticity field, while the energy transport equation describes its statistical behavior. Both consist of terms for advection, generation, dissipation and scale transfer, but have different advantages for a Lagrangian representation. While the vorticity equation is well suited for describing dynamics, the injection and dissipation of energy via particle creation and dissipation is easier in an energy formulation. We will use a combination of both representations to leverage the strengths of both models.

Motion equation The motion of vortex particles is described by the vorticity equation (4.20). However, we will use the vortex particles not as a full representation of the velocity field, but use it in combination with a grid-based Navier-Stokes solver. The velocity field $\mathbf{u}$ therefore consists of two parts, the flow field of the grid solver $\bar{\mathbf{U}}$ and the detail velocity induced by the vortex particles. We leave external forces and baroclinity to the underlying

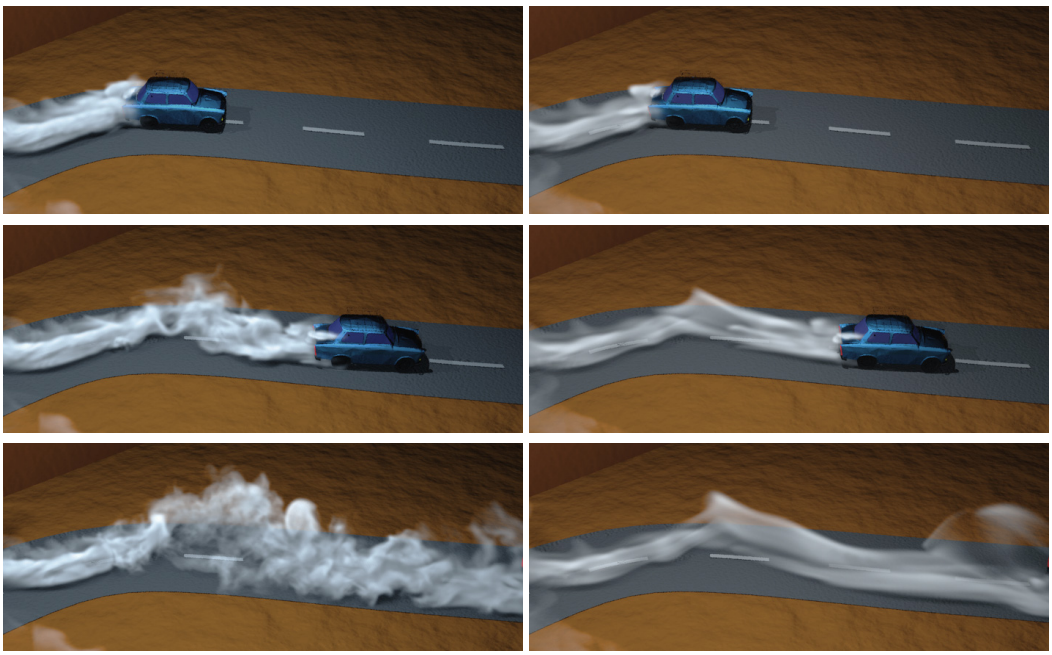


Figure 4.20: Example of a moving object inducing turbulence in its wake. On the left, the base simulation is shown. In the images on the right, this simulation has been augmented with vortex particles using our method. As the car accelerates (middle and lower picture), more turbulence is expected – this behavior is correctly predicted by our model.

solver, which will affect the vortex particles via its velocity field. With this, the evolution for the vortex particles becomes

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} \; . \tag{4.32}$$

The left side of the equation is handled by advecting the particles in the final high-res flow field augmented with turbulence. The first term on the right-hand side is the vortex stretching term. It is computed by trilinear interpolation of the discrete gradient of the velocity grid, and is used to adjust the particles' vorticity magnitude by $\Delta t(\boldsymbol{\omega} \cdot \nabla)\mathbf{u}$. This term is problematic as it might introduce exponential accumulation of vorticity magnitude in a particle. Therefore, the particle is rescaled after the update to preserve the magnitude, effectively only spinning the particle, but not altering its strength. The strength, effectively a measure of energy gain and loss, is handled by the energy dynamics, which is explained in the next section. Similarly, the viscous diffusion term (the second term on the right hand side of (4.32)), will be handled by energy dynamics, as it is not easily represented on a sparse particle system.

This gives us a reduced formulation of (4.32) which conserves vorticity as well as energy. It is therefore orthogonal to the energy transfer equations, the computation of which we will describe next.

Energy dynamics To model the transfer of energy, we will model energy transport in the sense of a turbulence model Section 2.2. As the turbulent energy in our system is

represented using vorticity, not kinetic energy k, it is not practical to directly solve the energy transport equation of a turbulence model. Instead, we express the individual energy transport terms to our vorticity model and apply it directly on our particle system. In its most general formulation, the turbulent energy transport equation states

$$\frac{\partial k}{\partial t} + (\mathbf{u} \cdot \nabla)k = -\nabla \cdot \mathbf{T} + \mathscr{P} - \varepsilon \quad . \tag{4.33}$$

The left hand side again is represented by advection of the particles. The right-hand side consists of production $\mathscr{P}$, dissipation ε and the energy transfer term $\nabla \cdot \mathbf{T}$, which is approximated using the gradient diffusion hypothesis in classical turbulence models. Outside the inertial subrange, this quantity is however hard to model. Its behavior will therefore be based on the length scale, as explained below.

For the production term, we can use the information from our artificial boundary layer (see Section 4.3.3). The dissipation ε occurs at wavenumbers that are usually well below the resolved grid resolutions. Dissipation is therefore implemented by removing particles whose radii are too small to be represented on the grid. We use a threshold of $2\Delta x$ for our simulations. Finally, for handling the remaining energy transfer term for $\mathbf{T}$ of (4.33), we distinguish the following two cases:

1. *The particle is in the inertial subrange*. We represent the energy cascade by decaying a particle with wavenumber κ_a into n particles of smaller wavenumber κ_b. We typically use n=2 in our simulations. From Kolmogorov's law, we can derive a timescale of decay as $\Delta t = C(\kappa_a^{-\frac{2}{3}} - \kappa_b^{-\frac{2}{3}})$, where C denotes a parameter that depends on the rate of dissipation ε. In practice we can use a value normalized by the averaged flow $\mathbf{U}$ here. We also know that for the turbulent energies $k_a/k_b = n(\kappa_a/\kappa_b)^{-\frac{5}{3}}$ holds, which is used to derive the vorticity magnitude of the new particles. For practical reasons, we also add a small position and angle displacement to the new vortex particles, as they would otherwise lump together.
2. *The particle is in the model-dependent range*. As transfer cannot be easily described in this regime, a heuristic is used. Typically, small vortices with aligned direction tend to form larger vortices in this range. Therefore, we merge vortex particles in the model-dependent range with a distance of less than the particle radius to a single larger vortex particle. Here, the vortex magnitude is chosen so that total the energy is conserved as $k_{new} = k_1 + k_2$. As very small and strong vortex particles might induce stability problems, we also conserve the energy density, i.e. $\frac{k_{new}}{V_{new}} = \frac{k_1}{V_1} + \frac{k_2}{V_2}$. This specifies the radius and strength of the merged particle. The new direction is obtained by a weighted average, with the respective energies as a weight.

Vorticity Synthesis

To synthesize turbulence from the vortex particles, we need to obtain a detail velocity field from the particles system. Each vortex particle has a vorticity vector $\boldsymbol{\omega}_P$, encoding magnitude and rotation axis, and a kernel over which this value is applied. The detail field can be obtained by integrating this vorticity kernel and mapping it on a higher-resolution grid.

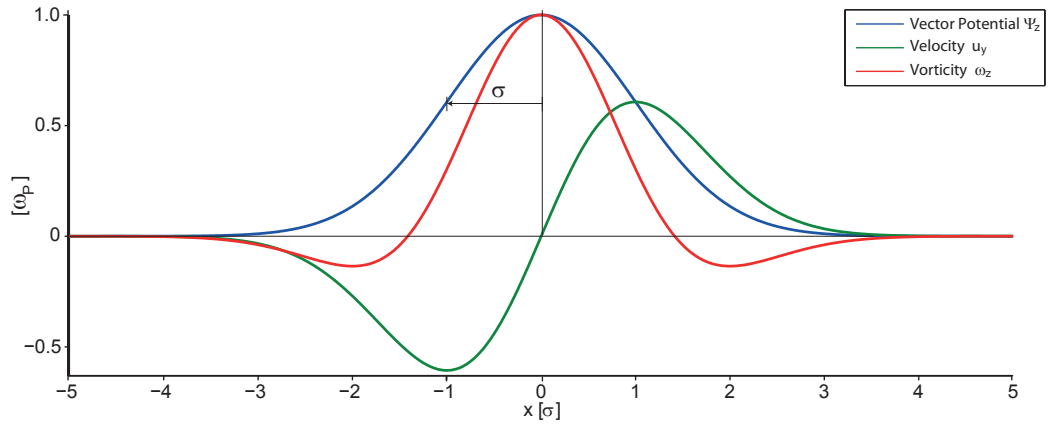


Figure 4.21: Vector potential, velocity and vorticity of the vortex particle kernel are shown along a x-axis slice.

Kernel For the direct regulation of vorticity, a kernel with the following properties is desired: at the vortex particle center, vorticity should be equal to $\boldsymbol{\omega}_P$. Also, the resulting vector field should mainly contain rotation around $\boldsymbol{\omega}_P$, and smoothly fade out with the particle radius without causing discontinuities. And lastly, the associated velocity field, and its integral, which is needed for e.g. energy calculation, should be a simple analytic form. We chose a Gaussian peak with standard deviation of σ in the vector potential to meet these requirements. In cylindrical coordinates, it is given by

$$\Psi(z,\varphi,\rho) = -|\boldsymbol{\omega}_P|\,\sigma^2 \exp^{\frac{-\rho^2-z^2}{2\sigma^2}} \mathbf{e}_z\,, \tag{4.34}$$

where the vortex axis $\mathbf{e}_z$ is aligned with $\boldsymbol{\omega}_P$. We can then derive the velocity field

$$\mathbf{u} = \nabla\times\Psi = -|\boldsymbol{\omega}_P|\,\rho \exp^{\frac{-\rho^2-z^2}{2\sigma^2}} \mathbf{e}_\varphi\,, \tag{4.35}$$

and the vorticity kernel

$$\boldsymbol{\omega} = \nabla\times\mathbf{u} = -\frac{|\boldsymbol{\omega}_P|}{\sigma^2}\left(\rho z\mathbf{e}_\varphi - (\rho^2-2\sigma^2)\mathbf{e}_z\right)\exp^{\frac{-\rho^2-z^2}{2\sigma^2}}\,. \tag{4.36}$$

A cut-off radius is used to make the kernel support finite. We use $r=\sqrt{6}\sigma$ at which point the exponential term of the kernel function has fallen to 10^{-3}. The length scale is defined at the kernels' origin, so that its wavenumber is $\kappa=\frac{1}{\sigma}$. For the contained energy $E\propto\boldsymbol{\omega}_P{}^2\,\sigma^5$ holds.

Synthesis To combine the detail field and the velocity field from the underlying solver, these two fields could be simply added, as in the method in Section 4.2. However, we also want to allow vortex sizes above the grid resolution of the underlying solver, which means turbulent detail might overlap with existing vortices from the base solver. Therefore, we need to exclude vorticity already represented in the base solver to avoid duplication. To achieve this, the base grid vorticity is measured, and only the difference is synthesized on the detail field.

The synthesis is a three-step process: First, the vorticity field $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ of the velocity grid is computed by finite differences. Second, all particle vorticity kernels are summed up to obtain a desired vorticity field $\boldsymbol{\omega}_D$. And third, each particle adds its kernel to the velocity field, scaled by a weight w_k, computed as:

$$w_k = \frac{\sum_{kernel}(\boldsymbol{\omega}_D - \boldsymbol{\omega}) \cdot \bar{\boldsymbol{\omega}}_P}{\sum_{kernel} \boldsymbol{\omega}_D \cdot \bar{\boldsymbol{\omega}}_P} \quad , \tag{4.37}$$

where $\bar{\boldsymbol{\omega}}_P$ is the particles' normalized rotation axis. The dot product with $\bar{\boldsymbol{\omega}}_P$ ensures that only the vortex particles' direction is considered, and the kernel is normalized by the sum of desired vorticity. We achieve an exact regulation of the vorticity sum under the kernel in one timestep by this process.

Vortex particle seeding As explained in 4.3.3, particles will be seeded in regions of high normalized Reynolds stress. Based on the probability $p_T(\mathrm{x})$ from (4.31), a particle is created at position $\mathbf{x}$.

All confined vorticity $\boldsymbol{\omega}_{ABL}$ within the particle's radius is removed from the artificial boundary layer, and the particle's strength and direction $\boldsymbol{\omega}_P$ are set such that the vorticity integrated over the kernel equals the removed vorticity sum. We choose the particle radius to be as large as possible without touching an object. We allow for radii up to a size r_{max} which is fully resolved by the main simulation (we have used a value of $r_{max} = 6\Delta x$ below).

The constant c_P in (4.31) controls the granularity of the seeding process. If set to a high value, confined vorticity is turned into free turbulence relatively quickly. This results in a large number of weaker particles near the object, which then merge to large vortices. On the other hand, if c_P is set to a high value, the artificial boundary layer plume can grow, and fewer, stronger particles form. With an appropriately chosen c_P, numerical cost can be kept low while avoiding popping artifacts that may occur if overly large particles are seeded. We use a c_p of $\approx 1-4$ in our simulations.

4.3.5 Implementation

In this section, details on our implementation of the precomputation step and the main simulation loop are provided.

Precomputation

For precomputing the artificial boundary layer, it is essential to resolve the mean flow around an object. This can be done using time-averaging over a long period of time with a standard solver, or using a RANS solver. As we are only interested in the velocities around the boundary layer, we have used a standard solver with an artificially increased viscosity in the form of a diffusion step for the velocities. Due to the increased viscous effect, it stabilizes quickly and an average over fewer frames can be used. We have found that using the more complex RANS or longtime-averaging does not pay off visually compared to this more efficient solution. After obtaining the averaged flow field, the boundary layer is calculated according to the pseudo-code (Fig. 4.19) and stored as a point set.

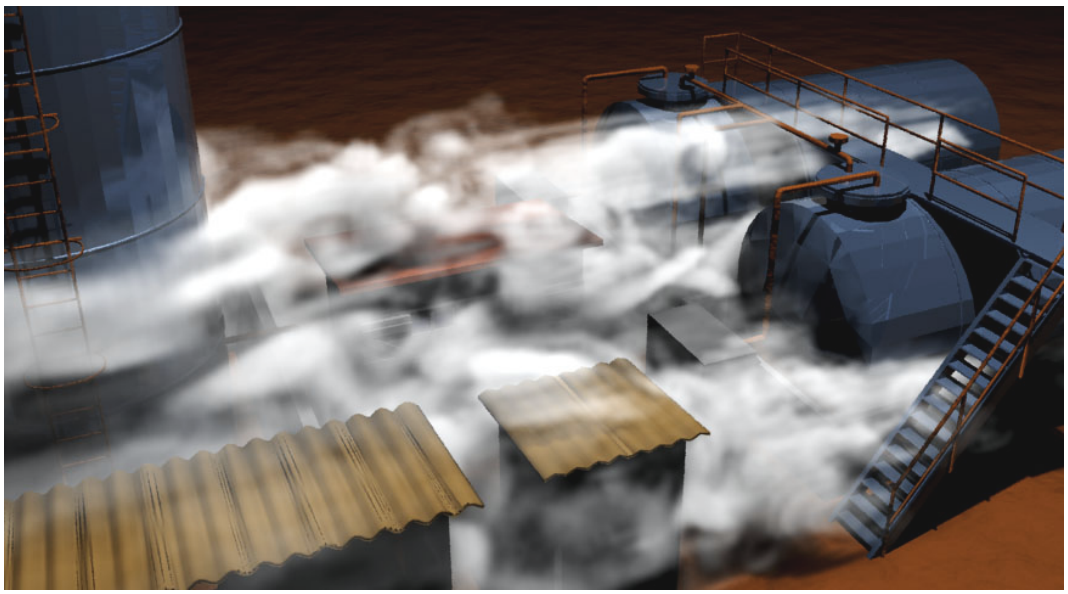


Figure 4.22: In this example a static flow field is used to generate complex turbulence around an object with our method.

Moving objects To precompute the flow for scenes with moving objects, the boundary layer around each moving objects is precalculated. If the object can move and rotate freely, or its movement is not known a priori, our algorithm allows us to precompute the whole range of movement directions to later on generate arbitrary simulations of the object in a flow. For this, we split the movement into a translational and a rotational component and precompute artificial boundary layers for each.

To perform the precomputation, the object is placed in the center of a simulation grid. The domain box is chosen large enough not to disturb the flow around the object. For the translational component, we leave the object fixed and use different inflow velocities, defined as boundary conditions on the domain box. As $\boldsymbol{\omega}_{ABL}$ is linear in the velocity magnitude, we only need to sample the velocity direction. In our simulations, we use 10×20 samples in spherical coordinates. For the rotational component, the object is placed in a standing fluid, and we rotate the object with normalized speed around a chosen axis. Again, we use 10×20 samples in spherical coordinates to sample the rotation axis direction.

The simulations stabilize quickly due to the increased viscosity. We have used 50 steps for the examples shown in our video. In the precomputations for the rotational component, this is equivalent to one full rotation. After stabilizing, we average the velocities over another 50 frames. We note that precomputations for each direction can be trivially done in parallel.

Applying the precomputed set At simulation time, we determine the objects linear velocity relative to the scene, and its rotation axis. We then look up the nearest values in the precomputed database. A bilinear spherical interpolation is performed for both the linear velocity direction as well as the rotation axis. The results of the interpolation are scaled by respective magnitude and added. We have performed error measurements for the linear in-

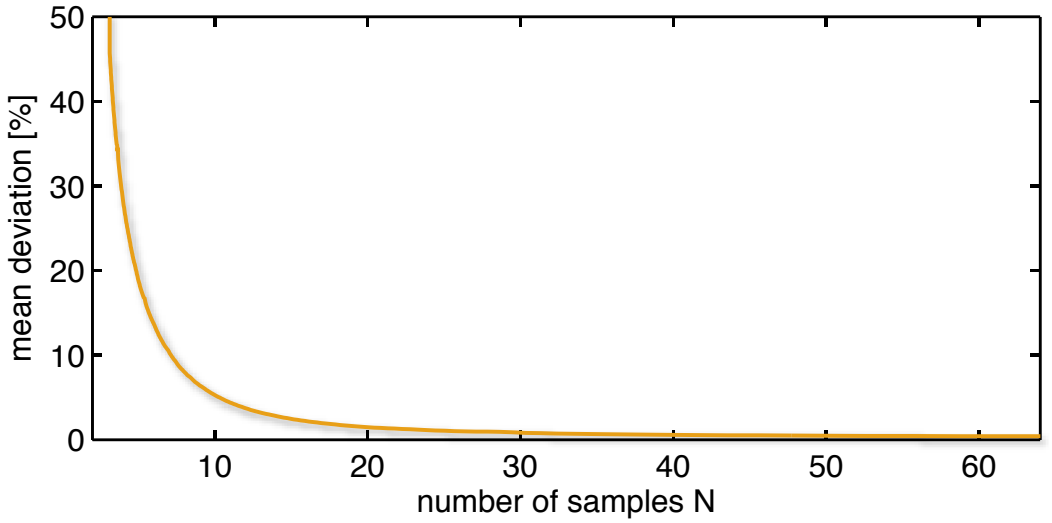


Figure 4.23: Mean relative error of the artificial boundary layer values for the car model. A reference simulation is compared to a spherical interpolation with N samples for the azimuth.

terpolation of the boundary layer values. The corresponding graph can be seen in Fig. 4.23. Our choice of 20 directional samples in the azimuth means we have an interpolation error of 1.6%. As the decomposition into rotational and linear component is only an approximation, we have also measured its error for the car model (Fig. 4.20). In this case the error is 8% on average, and thus small enough not to cause visual artifacts.

It is not necessary to fully resolve the boundary layer during the precomputations, as our model described in Section 4.3.3 takes care of this. Instead, one should make sure the resolution of the precomputation is sufficiently fine to resolve all important geometric features of the object. This is eased by the precomputation focusing only on that object, even if it will only occupy a tiny fraction of the final simulation domain. In addition, since the precomputation can be used on many simulations, a high resolution precomputation grid can quickly pay off. In Section 4.3.6 we demonstrate the effectiveness of the precomputation even when an object is extremely thin.

Simulation loop

For the actual simulation, a standard fluid solver and a vortex particle system are coupled. In each simulation step, the artificial boundary layer is updated, new vortex particles are created and vortex particle dynamics are applied. Afterwards, the turbulence forces are added to the flow field, and finally, the remaining steps of the standard fluid simulation are performed. This is repeated each time-step. Pseudo-code for this extended simulation loop can be found at the end of this section.

Note that we can also independently choose a higher grid resolution for the evaluation of the vortex particles. This allows us to more accurately evaluate the particle kernels, which is especially useful for small scale vortex details. This high resolution velocity field is down-sampled for the main simulation steps (line 28 in the pseudo-code), and up-sampled for our

Setup	Fig. 4.15	Fig. 4.20	Fig. 4.24	Fig. 4.22
Grid res.	160·70·160	150·40·200	100·25·60	250·80·150
ABL upscaling	2	2	2	1
Frame time [s]	19.5	13.4	10.0	1.3
ABL time [s]	5.5	3.7	0.05	0.7
# particles	∼900	∼700	∼600	∼1000
Vortex gain β	6.2	0.4	3.2	4.0
Precomp. res.	70x150x70	70x70x120	100x25x60	250x80x150
Precomp. [s]	220	112	59	227

Table 4.2: Detailed statistics for our simulation runs. *ABL upscaling* refers to the upsampled grid on which vortex particle evaluation and smoke/levelset advection is performed. The precomputation time is given per database parameter.

algorithm before starting with line 1. Performing the algorithm (line 1–25) with a higher resolution enables us to simulate detailed features, e.g., when advecting smoke densities or a free surface level set, while the costly pressure projection operates on a small grid resolution. Typically, we have used a two times higher resolution for the examples below.

4.3.6 Results and Discussion

In the following section we discuss comparisons of our method to previous work and a reference simulation. In addition, we demonstrate several complex examples of turbulence being generated around moving objects or due to effects such as wind or a flowing river.

Comparisons In Fig. 4.18 the effect of wall-induced turbulence can be seen for the flow over a cavity with a grid resolution of $120 \times 60 \times 40$. The top image shows a standard, unmodified simulation, while the lower image uses our algorithm to introduce wall-induced turbulence. Both simulations use the same grid resolution, but the unmodified simulation is unable to capture any turbulence being generated from the shearing near the walls. Our simulation exhibits complex vortices due to the vorticity generated at the wall boundaries. To compare our method to approaches for synthetic detail generation, we have simulated the same cavity setup including wavelet turbulence, using the implementation available on the paper's website [KTJG08]. This comparison is shown in Fig. 4.25(B). Wavelet turbulence successfully adds small detail to the overall flow, but has difficulties introducing larger vortices to the strong horizontal motion. In contrast, our method introduces persistent larger vortices, while the resulting smoke filaments are successfully broken up by the wavelet turbulence. This shows that our method is suitable for bridging the gap between small synthetic vortices and the vortices resolved by a standard simulation.

We use the setup shown in Fig. 4.25 to compare our method with normal vortex particles [SRF05]. The simulations now focus on the left edge of the cavity. The image (A) shows a reference simulation, using a four times increased grid resolution. Note that the flow along the wall to the left is completely straight, while turbulent structures form to the right of the backward facing step. This behavior has been confirmed in various experiments and simulations (e.g. [LMK97]). The image (C) shows the flow after randomly introducing

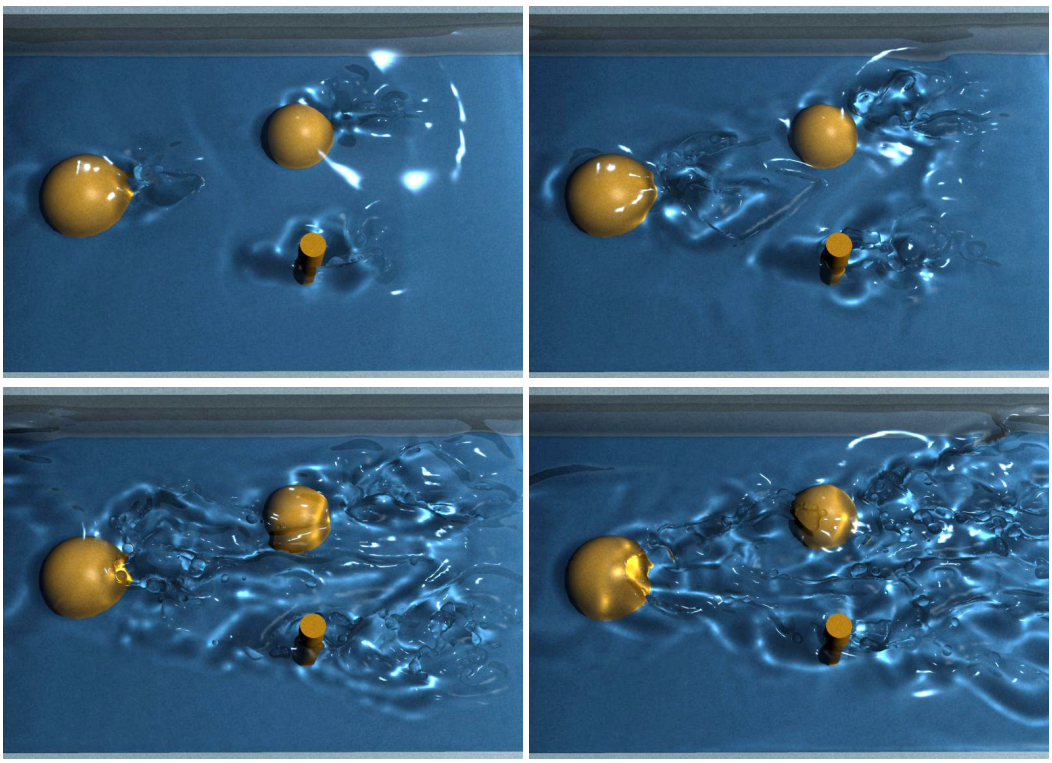


Figure 4.24: Our algorithm naturally extends to simulations of liquids. Here, we apply our algorithm to a river flow around three obstacles, resulting in turbulent wakes behind them.

vortex particles along the walls. Naturally, the vortex particles do not take the overall flow into account, and strongly distort the structure of the flow. Our method, shown in picture (D), is able to recover the vortices being shed off the step, without distorting the flow along the wall to the left. Although we are not able to fully recover the flow of the reference simulation due to the different numerical viscosities, our method is able to qualitatively capture the wall-induced turbulence at a much lower computational cost. On average, the time per frame for the reference simulation was 218 times higher than for the simulation with our algorithm.

The limitation that motivated vortex particles was that vorticity confinement uniformly amplified vorticity magnitude. Our method, like vortex particles, overcomes this by allowing local modeling of vorticity, including effects like tilting and stretching. However, by modeling the boundary layer and considering the directions of particles vortices with (4.26), we are able to keep vortex particles from disturbing the bulk flow. This allows us to use vortex particles of larger magnitude than the randomly seeded vortex particles.

Complex Examples Next we consider examples with more complex geometry. Fig. 4.20 shows the simulation of a moving car that is emitting smoke. It can be observed how our model reproduces the dependence of turbulence strength from the cars velocity. As can be seen in the top row of Fig. 4.20, a normal simulation of the same resolution would not re-

solve any shed vortices at the car's surface. Second, Fig. 4.15 shows a thin whisk geometry stirring smoke. The boundary layer precomputation was done with a high resolution grid that resolved the whisk's wires, while its subsequent use in a smoke simulation was done on a much coarser grid. The standard grid did not resolve the wires, and only approximate velocity boundary conditions were set, resulting in the fluid slightly following the whisk's motion. Still, our algorithm was able to accurately generate vortices that are produced by its motion.

In Fig. 4.22 we show how our method works in conjunction with static flow fields. In this case we precompute a snapshot image of static flow around the object, and use it to advect the boundary layer, the vortex particles and the smoke densities. This simple form of simulation works without an expensive pressure correction step. Despite the simple underlying setup, we are able to produce complex structures forming in the wake behind the obstacle from the interactions of the vortex particles amongst themselves. Lastly, we demonstrate that our method can be easily extended to free surfaces in Fig. 4.24. Here three obstacles in the liquid produce turbulent wakes behind them. For this simulation, a particle level set [EMF02] was used to represent the liquid's surface. Similar to particles near obstacle walls, we reduce a particle's kernel size once it extends past the liquid phase to avoid non-divergence free velocity fields.

Detailed grid sizes and timings for the examples above can be found in Table 4.2. The performance was measured on an Intel Core i7 CPU with 3.0 GHz. The majority of the time used for our approach (denoted by *ABL time* in Table 4.2) is taken up by the advection of the artificial boundary layer. For the liquid example of Fig. 4.24, the performance is strongly dominated by the particle level set. Overall, we achieve computing times ranging from 10 to 20 seconds per frame on average. An exception is the example with a static flow field, which requires only 1.3 seconds per frame.

4.3.7 Conclusions

In this chapter, we have presented an algorithm for simulating wall-induced turbulence. By leveraging turbulence modeling and wall flow theory, we are able to precompute turbulence generation based on the obstacle geometry. This precomputed object can then be included in various simulations. During the simulation, we determine transitioning regions and introduce appropriate vortex particles to represent turbulence. The particles are then evolved according to the vortex equations of flow to respect energy conservation and cascading. This yields the ability to efficiently compute physically plausible simulations of turbulence around rigid objects in a variety of settings. In contrast to other turbulence methods, the model presented can be used for obstacles which are too fine to be resolved on the simulation grid, and it can be applied for free surface flows, a topic that has been barely studied in previous work on turbulence. We do note however, that the model is passive in a sense that turbulence generation from the liquid surface is not handled in the model.

A limitation of our method is that our precomputation assumes a rigid object, making it difficult to apply it to deforming objects such as cloth. To resolve this, a RANS solver could be coupled to a normal fluid solver to determine the current shear stresses at the object surface. Alternatively, it may be possible to precompute suitable boundary layer data for deforming objects by making use of data compression schemes. Also, the approach for the

precomputation assumes the flow around the object can be described by the translational and rotational velocity components. If the flow around the object varies strongly, e.g., due to strong external forces or due to multiple objects in close vicinity, the resulting confined vorticity can differ from the desired values. It should be possible to replace the precomputation by a more powerful turbulence model; this would however lose the advantage of subgrid-accurate treament of obstacle turbulence.

As we modeled the turbulence generation based on wall flow, the method will only generate turbulence from obstacle interaction. Turbulence driven by free-stream effects or buoyancy must be handled using other methods. In addition, a trade-off of our method is the use of vorticity reconstruction at a higher resolution. While this allows us to go beyond the coarse simulation Nyquist limit and get higher resolution detail (a limitation of the original vortex particle method), it means the domain and object boundaries as well as the free-surface boundary conditions are not as well modeled by the reconstructed high resolution velocity field. The need for a high-resolution field also limits the detail level that can be achieved, as for big scenes with fine detail, memory and computation time for operations on the high-resolution field can easily become the bottleneck.

The method presented in this chapter is specifically designed for the important special case of obstacle-induced turbulence. Many flows, such as plumes or explosions, however involve expansion forces and buoyancy, which are also strong sources of turbulence. In the following chapter, we will introduce a method that is able to represent these free-stream sources of turbulence, but can also handle obstacle source. Using vortex methods on interfaces, we can even avoid the need for a high-resolution grid to represent detail.

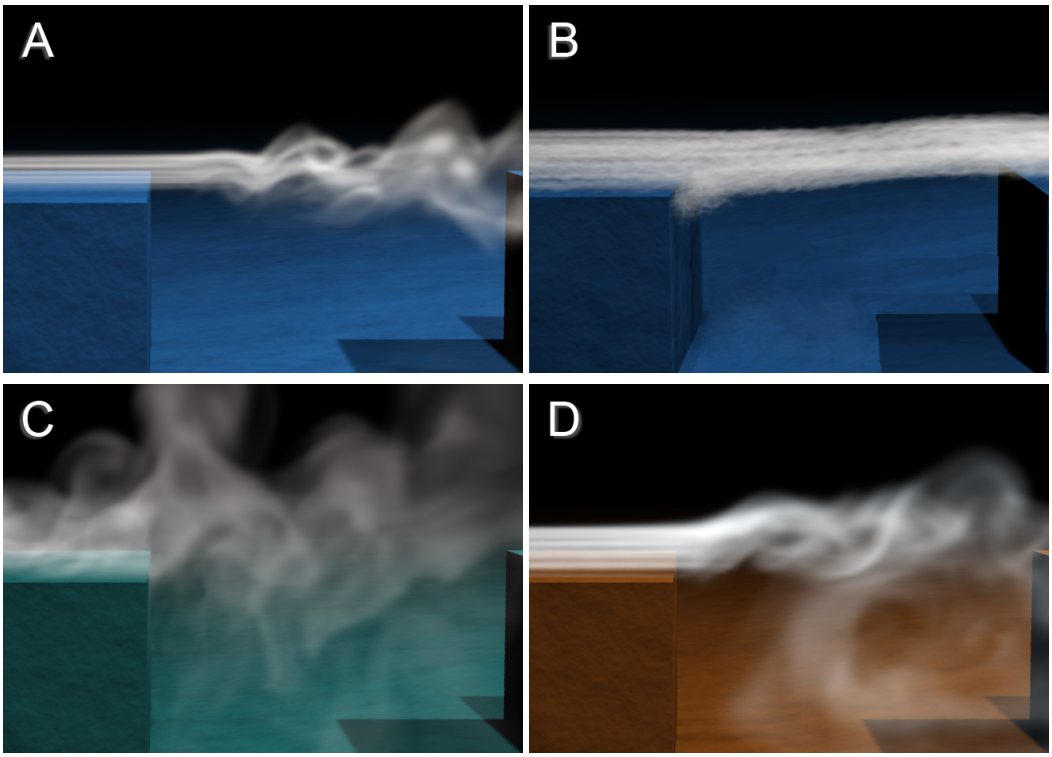


Figure 4.25: Comparison between a high-resolution reference simulation (A), Wavelet turbulence (B), a simulation with randomly seeded vortex particles along the walls (C), and our method (D). Wavelet turbulence does not predict the turbulence formation, and therefore only amplifies noise. The random vortex particles destroy the overall flow structure, although the number and strength of the vortex particles are similar to those used in our method. Our approach correctly identifies the turbulence being shed off the step, similar to the reference simulation (A), and insert particles into the flow with the correct orientation, strength and seeding position.

```
1:  // Initialize boundary layer
2:  for each voxel x on an obstacle boundary do
3:      Find corresponding (x_pre, ω_pre) in precomputed set
4:      // Initialize wall-attached ABL
5:      ω_ABL(x) ← max(ω_ABL(x), ω_pre)
6:  end for
7:
8:  // Simulate boundary layer separation
9:  Advect ω_ABL with the main flow
10:
11: // Seed vortex particles
```

```
12: for each voxel x with ω_ABL(x) ≠ 0 do
13:     p_T ← 2 c_P Δt (l_m |ω_ABL(x)|/|U_0|)^2
14:     if random() < p_T then
15:         (Y) ← voxels within particle radius of x
16:         ω_S = Σ_(Y) ω_ABL    // sum within particle radius
17:         ω_ABL(Y) ← 0         // remove vorticity from ABL
18:         Seed particle at x with total vorticity ω_S
19:     end if
20: end for
21:
22: // Vortex particle dynamics
23: Advect vortex particles
24: Merge, split, dissipate vortex particles (Section 4.3.4)
25: Synthesize turbulence (Section 4.3.4)
26:
27: // Standard fluid simulation steps
28: Velocity self-advection, pressure projection etc.
```

Pseudo-code for a main simulation loop including our turbulence model.

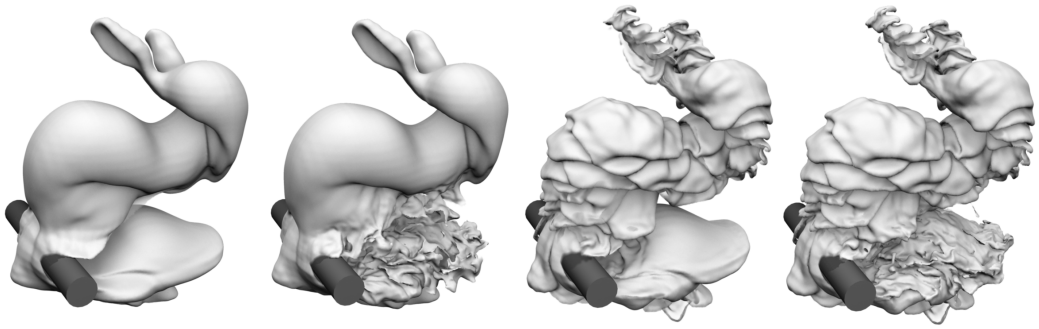


Figure 4.26: A dense cloud subject to buoyancy forces and interaction with a moving obstacle is simulated. We use a Eulerian solver to compute a base flow, as shown on the left. Small-scale detail is synthesized directly on the interface of the cloud. An adapted turbulence model provides details from obstacle interaction (middle left), while small-scale buoyancy effects are calculated using vortex sheet dynamics (in the middle right). The picture on the right shows the combined model.

4.4 Buoyant Turbulence

One of the most visually interesting features of turbulent flows is their complexity. Smoke plumes from volcanoes, explosions or collapsing buildings show detailed motion on scales from several meters down to the millimeter range, and the structure of the developing turbulent eddies is clearly visible at the sharp interface of the thick smoke and the air. At the same time, the thick clouds typically hide everything that is happening further inside the volume. Unfortunately, such scenes are numerically expensive to simulate, and we spend large amounts of computation on detail inside the cloud that will never be visible.

One way of dealing with these complex flow are volumetric turbulence methods, as presented in Section 4.2 and Section 4.3. However, even with a turbulence model the synthesized detail has to be represented in the simulation, and using a volumetric representation resolving the small-scale details requires immense storage capacity.

In this chapter, we introduce the vortex sheet method by Pfaff et al. [PTG12] to solve this problem. It explicitly discretize and track only the smoke-air interface, and greatly reduces the amount of information we need to store. In addition, this representation is a very suitable basis for detail synthesis. Instead of unnecessarily calculating detail that is hidden inside the smoke volume, we restrict synthesizing detail purely to the visible smoke interface.

The phenomena mentioned above exhibit another interesting effect: turbulence production in such flows mainly stems from buoyancy, which induces a vortex sheet at the smoke-air interface. This sheet reinforces small-scale surface instabilities, which then develop into turbulence. This means that the transition region where the turbulence is created is clearly visible, and this turbulent onset strongly influences the visible shape of the interface. However, the simulation resolution is typically too limited to directly capture these small-scale buoyancy effects. Furthermore, most turbulence models assume fully-developed homogeneous turbulence, which means they are valid inside the bulk smoke volume, but not at

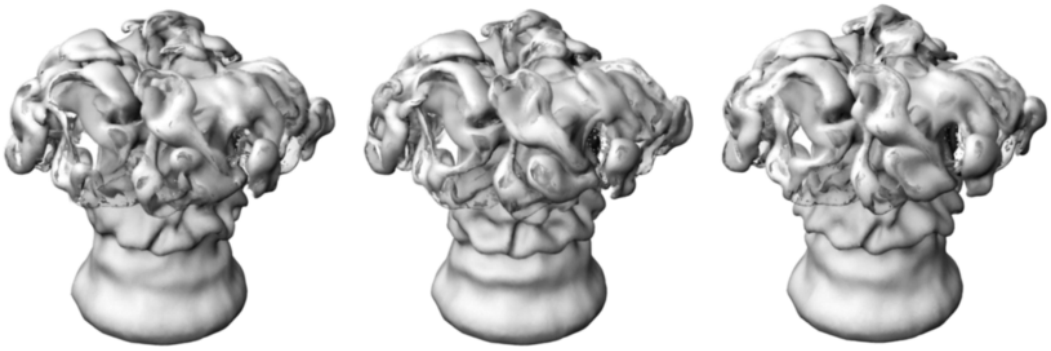


Figure 4.27: A buoyant plume is simulated without evaluation cutoff (left), with a cutoff of 10 cells (middle) and 5 cells (right, our default setting). While details are different due to accumulation of small differences over time, the visual quality is comparable.

the interface. Here, the turbulence generation process is highly anisotropic and model-dependent in nature. This means it is not well described using the statistical approaches that are the basis for most turbulence methods.

Our method addresses this problem by directly tracking the vortex sheet at the smoke-air interface. This allows us to compute buoyancy effects at scales independent of an underlying grid, and accurately model the turbulence generation process due to buoyancy. While vorticity-based methods are well-suited to describe turbulence formation, correct handling of obstacle boundaries is very difficult. The model therefore handles basic interaction with static or moving obstacles using a Eulerian solver, and tracks the obstacle-induced turbulence with a model specifically tailored to our needs. The turbulence model for obstacles is orthogonal to the buoyancy approach, which makes it possible to use both in combination or separately as needed. We use an adaptive triangle mesh to simulate non-diffusive smoke surfaces, and couple it to an Eulerian solver which captures the large-scale motion of the flow.

4.4.1 Vortex primitives

Fluid solvers in graphics typically use the velocity formulation of the NS equations to obtain the fluid motion. For dealing with turbulence, however, the vorticity formulation of the NS equations is often advantageous. Lagrangian solutions for the vorticity equation are common in CFD, and becoming increasingly popular in Computer Graphics. This thesis is heavily based on Lagrangrian methods for vorticity, therefore the theory is expanded here in more detail.

There are two ways in which Lagrangian vortex elements can be used. On the one hand, they can be used to discretize the complete vorticity field as an alternative representation to velocity. On the other hand, sparse elements can be used to augment an existing simulation. The second approach can increase performance by focusing resolution in desired areas. Also, remeshing inaccuracies are not as critical since the underlying simulation provides consistency. However, the interplay between the vortex primitives and the underlying simulation is nontrivial. In particular, vorticity structures need to be mutually exclusive in

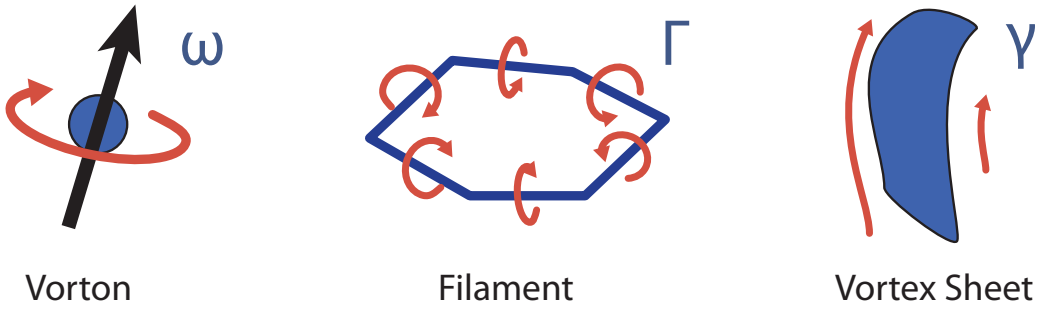


Figure 4.28: Three different Lagrangian primitives to represent vorticity are shown, with their induced velocity marked in red. Vortons are particles which induce a rotation around the axis of their associated vorticity $\boldsymbol{\omega}$. One-dimensional curve primitives are called filaments. They induce circular motion around the curve tangent based on a circulation number Γ. Vortex sheets contain the vorticity γ confined to a surface. They represent a velocity jump between two flow regimes.

both simulations, to avoid injecting excess energy into the system.

Vortons, as introduced in Section 4.3, are one primitive for discretizing vorticity. Below, we will introduce two further types. While *Vortons* are zero-dimensional point elements, *Filaments* represent vorticity confined to a one-dimensional curve, and *Vortex sheets* describe vorticity on a thin two-dimensional surface. Fig. 4.28 visualizes these primitives. In theory, a given vorticity field can be discretized by any of the three primitives. However, each of them has inherent advantages and disadvantages as far as re-meshing, motion equation and connectivity are concerned. Also, some forms allow a more natural representation of a certain flow geometry than others. It is e.g. possible to discretize vorticity on a thin surface using a patch of filaments. However, it is much easier to ensure a uniform coverage of a surface deforming in the flow by representing it using vortex sheets. A detailed comparison of vortex primitives can be found in [Sto06] and [CK99].

Filaments

Vortex filaments discretize vorticity using one-dimensional line segments or spline curves. Instead of directly storing the contained vorticity on the line elements, an equivalent representation is used. This has advantages for the formulation of the motion equation, as will be shown is the next paragraph. Each segment has an associated circulation number Γ which defines a rotation around the curve segment. It relates to vorticity by

$$\boldsymbol{\omega}(\mathbf{x}) = \Gamma(s)\,\mathbf{t}\,\delta(\mathbf{r}_\perp(s)) \tag{4.38}$$

where $\mathbf{t}$ denotes the line tangent and $\mathbf{r}_\perp$ is the distance perpendicular to the tangent of the curve. Filaments are suitable for e.g. rising smoke with low levels of turbulence, as the characteristic vortex rings arising in such settings are represented naturally using closed filaments.

By expressing the vorticity equation in the circulation formulation, we obtain

$$\frac{\mathrm{D}\Gamma}{\mathrm{d}t} = \nu \int_L \nabla^2 \mathbf{u} \cdot \mathrm{d}\mathbf{x} + \frac{1}{\rho} \int_L \nabla p \mathrm{d}\mathbf{x} \tag{4.39}$$

with the line integral of diffusion and baroclinity, respectively. Diffusion is hard to express efficiently for filaments, therefore filaments are mainly used in low-viscous flows, where diffusion can be neglected. This means that for flows without baroclinic generation, circulation effectively remains constant,

$$\frac{\mathrm{D}\Gamma}{\mathrm{d}t} = 0 \tag{4.40}$$

as vortex stretching is implicitly handled by elongation of the primitive itself. This is a very desirable property, as it avoids the problems associated with evaluation of the velocity gradient for vortex stretching in e.g. vorton methods. On the other hand, elongation also leads to ill-shaped line segments, therefore remeshing is necessary. A simple remeshing by subdividing elements that are too long or are strongly curved is often sufficient [Cho81]. For turbulent flows, however, this means that geometry will increase over time, as complex structures tend to generate even more complex structures on neighboring filaments. For fully-developed turbulence for example, filament strands will inevitably overlay and form a complex intertwined structure which could be represented using a much smaller number of vortons. Some effects of this can be mitigated by hairpin removal techniques [Cho96] or vortex loop optimization [WP10].

Filaments are integrated by directly evaluating the Biot-Savart law, which for circulation takes the form of the line integral

$$\mathbf{u}_{\blacksquare}(\mathbf{x}) = \frac{1}{4\pi} \int_L \Gamma(s)\mathbf{t} \times \frac{\mathbf{x} - \mathbf{r}(s)}{|\mathbf{x} - \mathbf{r}(s)|^3} \mathrm{d}s \quad . \tag{4.41}$$

In this form, $\mathbf{r}(s)$ is the line parameterization and $\mathbf{t}$ is the tangent. Regularization is introduced in the same manner as in (4.24). Depending on the concrete form of line discretization, flat line segments or spline curves, the discretized form of (4.41) varies.

Vortex sheets

The description of vorticity confined to a two-dimensional surface is a called a vortex sheet. Vortex sheets are particularly useful to describe thin sheets of vorticity that are induced at flow boundaries in the *boundary layer* and across steep density gradients, between e.g. cold and hot air of a rising plume. This surface is described by a surface mesh, triangle meshes being the most common approach. Vorticity on a surface element is represented using the *vortex sheet strength* $\boldsymbol{\gamma}$, which is defined as

$$\boldsymbol{\omega}(\mathbf{x}) = \boldsymbol{\gamma}(\mathbf{x})\delta(\mathbf{x}) \quad . \tag{4.42}$$

For deriving relations for the vortex strength, it is useful to consider the velocity jump $\Delta\mathbf{u}$ that is induced by the vortex sheet. It relates to vortex strength by $\boldsymbol{\gamma} = \mathbf{n} \times \Delta\mathbf{u}$, with the surface normal $\mathbf{n}$.

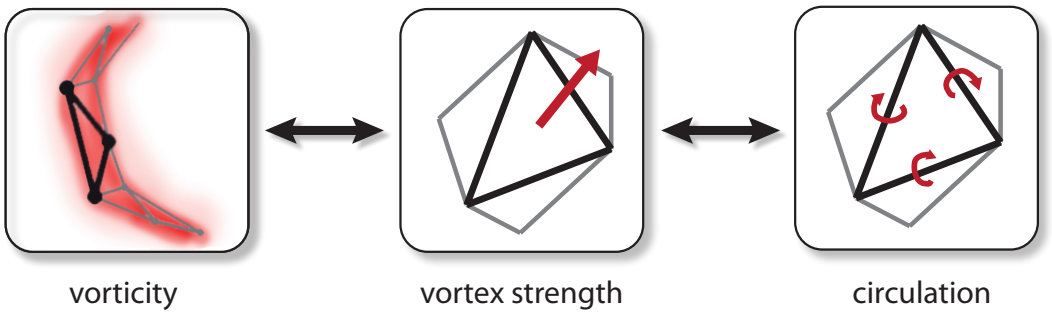


Figure 4.29: The continuous vorticity field around a surface can be represented in terms of a vortex sheet strength or circulation. Both are stored per surface triangle, and are equivalent representations. Vortex strength is a vector value, while circulation consists of three scalar rotation values around the edges of the triangle.

Using the velocity jump definition, the inviscid transport equation for vortex sheet strength can be derived as

$$\frac{\mathrm{D}\boldsymbol{\gamma}}{\mathrm{d}t} = \boldsymbol{\gamma}\cdot\nabla\mathbf{u} - \boldsymbol{\gamma}(\mathbf{P}\cdot\nabla\cdot\mathbf{u}) - 2\beta_A\,\mathbf{n}\times\mathbf{g} \quad . \tag{4.43}$$

The first term on the right-hand side is the familiar vortex stretching, while the second term describes changes in vortex strength due to elongation in the direction of $\boldsymbol{\gamma}$. Here, $\mathbf{P} = \mathbf{I} - \mathbf{nn}$ is the tangential projection operator. The baroclinity term is expressed using the Boussinesq approximation [Men78] which is proportional to the Atwood ratio β_A. The Atwood ratio relates the densities of the two fluids to each other, and is defined as

$$\beta_A = \frac{\rho_1 - \rho_2}{\rho_1 + \rho_2} \quad . \tag{4.44}$$

The Boussinesq approximation assumes a small Atwood ratio, and is valid for e.g. hot/cold air, but not air/water interfaces. As for filaments, diffusion is not easily modeled for vortex sheets, so they are mostly used in the inviscid limit. Re-meshing is essential for vortex sheet methods as the induced vorticity quickly deforms the sheets. As in the case of filaments, however, the splitting of ill-shaped elements causes a steady increase in geometry for turbulent flows.

The Biot-Savart law for the vortex sheets has the form

$$\mathbf{u}(\mathbf{x}) = \frac{1}{4\pi}\int_S \boldsymbol{\gamma}(\mathbf{x}')\times\frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3}d\mathbf{x}' \quad . \tag{4.45}$$

Again, regularization can be performed as in the case of Vortons. Alternatively, VIC can be used for integration and regularization.

Conversion between representations

The primitives introduced above use different representations for vorticity, namely vorticity $\boldsymbol{\omega}$, circulation Γ and vortex sheet strength $\boldsymbol{\gamma}$. As each representation has different properties

as far as dynamics or source terms are concerned, it is sometimes advantageous to convert between representations, and use the one most appropriate for the task. We will focus on vorticity confined to thin sheets here, as this theory is used in the presented method.

While vortex sheet strength, discretized using a triangle mesh, is the most natural representation for this case, its dynamic equations are more involved than those of the circulation formulation. According to Stock et al. [SDT08], the vortex sheet strength vector $\boldsymbol{\gamma}$ of a triangle uniquely relates to the three circulations numbers Γ_i around the triangles edge vectors $\mathbf{e}_i$. We can therefore express the motion equations in terms of both circulation and vortex sheet strength. This is illustrated in Fig. 4.29. It should be noted that the circulation numbers are defined per triangle, which means that adjacent triangles may have different circulation numbers for the same edge. In order to convert to vortex sheet strength, we can use the relation

$$\boldsymbol{\gamma} = \frac{1}{A} \sum_{i=1}^{3} \Gamma_i \mathbf{e}_i \tag{4.46}$$

with A denoting triangle area. On the other hand, conversion from vortex sheet strength to circulation can be performed by solving the overdetermined linear system

$$\begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{pmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \end{pmatrix} = A \begin{pmatrix} \boldsymbol{\gamma} \\ 0 \end{pmatrix} \quad . \tag{4.47}$$

During this conversion process, the vorticity component normal to the surface is lost. For vortex sheets, this is however a desired property [SDT08].

4.4.2 Vortex Sheet Methods

In general, the evolution of vorticity can be described with the vorticity equation (4.20). For this method, we will focus on plumes with a sharp density interface, which is a good approximation model for e.g. heavy smoke plumes, or two liquids with different densities. Under the influence of buoyancy or external forces, a thin sheet of vorticity forms at this interface. In our model, we will not track the volumetric velocity or vorticity field, but represent this interface vorticity, or *vortex sheet*, on a surface mesh.

Most commonly, the vorticity in vortex sheets is expressed via the vortex sheet strength vector $\boldsymbol{\gamma}$. If we formulate the vorticity equation using $\boldsymbol{\gamma}$, we obtain the evolution (4.43), with terms for advection, vortex stretching, elongation and baroclinity. By integrating this equation we would be able to calculate the full dynamics of a buoyant plume. As the evolution equation contains operators which are hard to express on a surface representation, this is however not trivial. We therefore also make use of another expression of vorticity, namely the circulation. For vortex sheets, these representations are equivalent and can be converted as explained in Section 4.4.1. As some operations are formulated easier in a circulation notation than for vortex strengths, and vice versa, we can simplify the evolution equations by switching between representations. This procedure will be explained in the following paragraph.

Our Model To solve the vorticity dynamics equations, it is necessary to have a discretization of the interface. For this we use a mesh consisting of triangles, where each triangle i

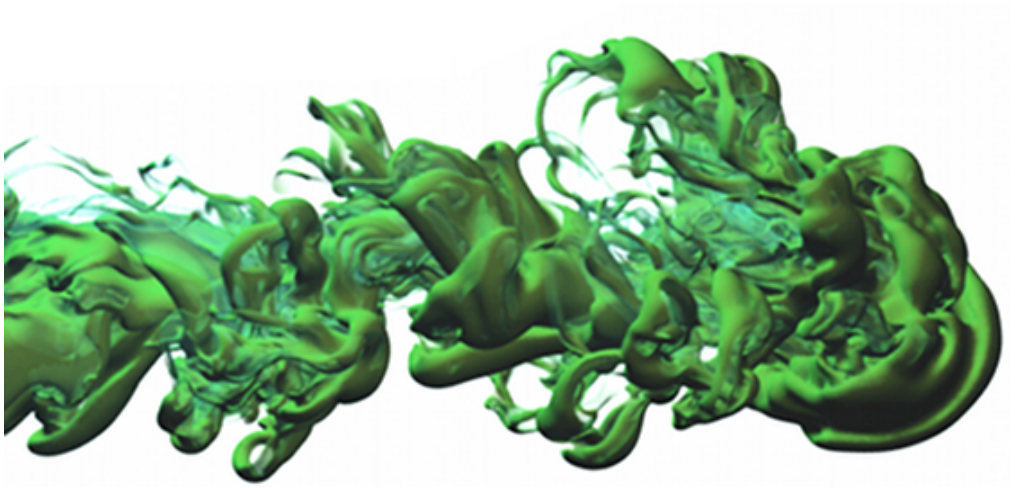


Figure 4.30: We simulate the dynamics of a dense fluid in water with pulsed inflow conditions. The buoyancy leads to complex surfaces in the downstream region to the right.

has a corresponding vortex strength $\boldsymbol{\gamma}_i$. As we want to make use of the filament representation, too, the three circulation numbers are stored for each triangle in addition to the vortex strength. These circulation numbers $\Gamma_{1\ldots3}$ define a rotation around the triangle's edges $\mathbf{e}_{1\ldots3}$.

As we are interested in buoyant effects, we apply the baroclinic source term in vortex sheet strength notation for each time step.

$$\frac{\partial \boldsymbol{\gamma}}{\partial t} = -2\beta_A \hat{\mathbf{n}} \times \mathbf{g} \quad . \tag{4.48}$$

We now use the fact that the vortex stretching and elongation terms of the evolution equation are implicitly handled in circulation notation, and vanish from the equation. Before evaluating the advection of out surface mesh, we therefore switch to circulation notation (4.47) and return to vortex strength notation afterwards (4.46) Using this process, we can avoid the calculation of these operators altogether. We now have taken care of all terms in the vorticity equation. For evaluation of the advection term, we however still need velocity information. This can be integrated from the vortex strength values using the Biot-Savart law, as explained in Section 4.4.1. If we look at the integration equation

$$\mathbf{u}(\mathbf{x}) = \frac{1}{4\pi} \int \boldsymbol{\gamma}(\mathbf{x}') \times \frac{\mathbf{x}-\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|^3} d\mathbf{x}' \quad . \tag{4.49}$$

we however note that such an evaluation is numerically very complex. For each mesh node, we need to integrate over all triangles, which results in $\mathcal{O}(n^2)$ complexity. As we want to simulate very detailed meshes with millions of triangles, this is prohibitively expensive. Even more importantly, we note that so far, we are only able to obtain the dynamics prescribed by ideal buoyancy in free space. Most practical scenes, however, have a nontrivial underlying flow due to interaction with obstacles and boundary conditions. In the next section, we therefore introduce a local evaluation scheme, which resolves this issues.

// Grid-based Fluid solver
Semi-Lagrangian density and velocity advection
Add grid-based buoyancy
Pressure projection

// Turbulence model
Compute production: $\mathscr{P}_{wall} = 2\nu_T|\nabla \times \mathbf{U} - \boldsymbol{\omega}_g|^2$
Update $\boldsymbol{\omega}_g$ based on (4.56) and advect
Update k, ε based on (4.53) and advect

// Mesh dynamics
Integrate baroclinity: $\boldsymbol{\gamma}_i \leftarrow \boldsymbol{\gamma}_i - \Delta t\, 2\beta_A\, \hat{\mathbf{n}} \times \mathbf{g}$
Compute Gaussian filtered vortex strengths $\bar{\boldsymbol{\gamma}}_i$
Small-scale vortex strength: $\boldsymbol{\gamma}'_i \leftarrow \boldsymbol{\gamma}_i - \bar{\boldsymbol{\gamma}}_i$

Compute circulations $\Gamma_i \Leftarrow \boldsymbol{\gamma}_i$, (4.47)
for each mesh vertex i **do**
    $\mathbf{u}_i \Leftarrow$ Integrate (4.52) for sources $\boldsymbol{\gamma}'_i$ within r_C
    Advect vertex with $\mathbf{u}_i$ and grid velocity field
    Advect vertex with synthesized curl noise $\mathbf{u}_T = \sqrt{\alpha_S k}\mathbf{y}$
end for
Compute Vortex strengths $\boldsymbol{\gamma}_i \Leftarrow \Gamma_i$, (4.46)

Perform mesh surface smoothing
Perform edge collapses and triangle subdivision

Figure 4.31: Pseudo-code for the simulation loop of our algorithm.

Local evaluation

In the local evaluation model, we split the simulation into two parts: first, a Eulerian solver which computes a consistent flow field from obstacle interaction, inflows, and the large-scale effects of buoyancy. Second, a surface mesh which is used for front tracking of the smoke cloud and the simulation of detail due to small-scale buoyancy effects and obstacle turbulence.

For computation of the large-scale flow, we use a standard grid-based solver [Sta99] with second order semi-Lagrangian advection as described in Selle et al. [SFK+08]. Our vortex sheet approach enables us to use low grid resolutions, as details will be computed directly on the Lagrangian mesh. In the grid-based solver, a density field is tracked which is then used to compute coarse-scale buoyancy forces on the velocity field.

Evaluation of the small-scale buoyancy effects is performed using the vorticity of the mesh. To avoid duplication of buoyancy forces between grid and mesh, we remove the large-scale component of the baroclinic vorticity from the mesh. We first apply a Gaussian smoothing kernel on the vortex sheet strength $\boldsymbol{\gamma}$. The kernel width σ is set to match the grid cell width Δx to obtain the smoothed, grid-scale vortex strength component $\bar{\boldsymbol{\gamma}}$. The

difference $\boldsymbol{\gamma}' = \boldsymbol{\gamma} - \bar{\boldsymbol{\gamma}}$ now represents the details below grid scale, which are evaluated on the mesh.

By removing the mean only the high-frequency variations $\boldsymbol{\gamma}'$ remain, whose effect decays very quickly in the far field. This corresponds to the formation of small vortices, which act locally. We are therefore able to introduce a cutoff radius r_C to the evaluation. Only triangles within this radius have to be evaluated in the summation of (4.52). As we can rely on the grid solver to capture the large scale buoyant motion, the effects of this approximation are negligible. A comparison of a full evaluation versus two different cutoff radii can be seen in Fig. 4.27. As the cutoff approximation introduces small differences which accumulate over time, the resulting surfaces differ. However, the visual quality is comparable for all three simulations, while the processing time is five times faster using $r_C = 5\Delta x$. We use this value for all following simulations with our model. The position update for the mesh nodes is performed based on the Eulerian velocity field, and by applying a per-node velocity update for the small-scale structures, which is described next. The complete simulation loop for our combined solver is summarized in pseudo code in Fig. 4.31.

Regularization

To obtain the small-scale velocity update for the mesh, (4.49) is discretized, using the residual vorticity $\boldsymbol{\gamma}'$ as a source. As this equation is singular for points on the interface, we chose to regularize the equation analogous to the vortex blob regularization for vorticity particles [CB73]

$$\mathbf{u}_{reg}(\mathbf{x}) = \frac{1}{4\pi}\int_S \boldsymbol{\gamma}'(\mathbf{x}') \times f_{reg}(\mathbf{x}-\mathbf{x}')d\mathbf{x}' \tag{4.50}$$

$$f_{reg}(\mathbf{r}) = \frac{\mathbf{r}}{(|\mathbf{r}|^2+\alpha_R^2)^{\frac{3}{2}}} \quad . \tag{4.51}$$

The regularization parameter α_R effectively controls the minimal size of the generated vortices. We therefore set α_R proportional to the mesh resolution, as will be explained in Section 4.4.4. To discretize this equation, we use Gaussian quadrature. If $G_j(\mathbf{r})$ is the Gaussian quadrature of f_{reg} for triangle j, (4.50) becomes

$$\mathbf{u}_i = \frac{1}{4\pi}\sum_{j=1}^{m} A_j \boldsymbol{\gamma}'_j \times G_j(\mathbf{r}_i) \quad , \tag{4.52}$$

for all triangles within the cutoff radius. This means we need to evaluate a sum over all triangles $j = 1 \ldots m$ per mesh node i. In our examples, we use three-point quadrature, and refer the reader to [Cow73] for details on how to compute the integration weights.

4.4.3 Wall-based Turbulence Model

The method presented so is able to model buoyancy driven below grid scale, but does not deal with interaction with flow obstacles yet. While the coarse grid solver introduced in the local evaluation scheme provides the large scale interaction of the flow with obstacles, turbulence shed from these interactions is not represented. However, since our mesh representation allows us to evaluate synthesized turbulence directly on the interface, we can employ a turbulence model similar to Section 4.2 for this type of turbulence.

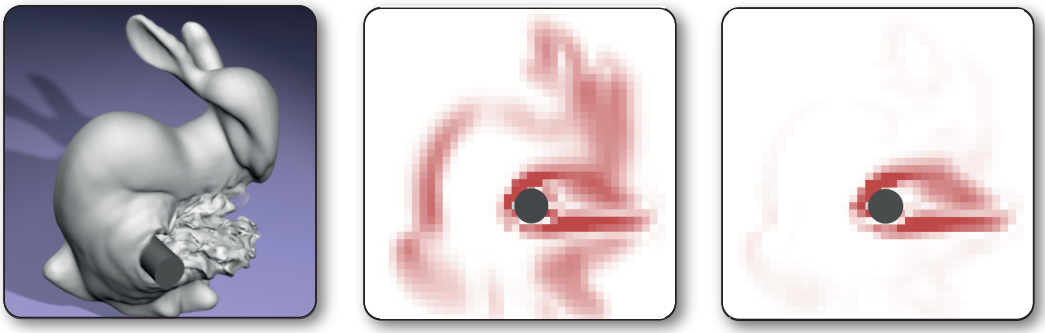


Figure 4.32: To separate the sources of buoyancy and wall-based turbulence, buoyant vorticity is tracked over time. The total vorticity of a snapshot from Fig. 4.26 is shown in the middle picture, while the difference to the tracked buoyant vorticity is shown to the right. The gray circle marks the position of the cylinder. We observe that despite a small residual halo, our model tracks the area of obstacle influence behind the cylinder very well.

The turbulence model we propose in the following is orthogonal to the buoyancy model of the previous sections, and both models can be used independently or in combination. We first model the spatial and temporal distribution of turbulent kinetic energy k using an *energy transfer model*, and then synthesize turbulent detail on the surface using *frequency-matched curl noise*. Below, we will briefly outline the theory used, and explain our modifications. Turbulence modeling is described in more detail in the Section 2.2.

Modified Energy Model

We compute the energy dynamics based on the commonly used k–ε model by Launder and Sharma [LS74], which models the evolution of the turbulent energy k:

$$\begin{aligned} \frac{\mathrm{D}k}{\mathrm{d}t} &= \nabla(\frac{\nu_T}{\sigma_k}\nabla k) + \mathscr{P}_{wall} - \varepsilon \\ \frac{\mathrm{D}\varepsilon}{\mathrm{d}t} &= \nabla(\frac{\nu_T}{\sigma_\varepsilon}\nabla \varepsilon) + \frac{\varepsilon}{k}(C_1\mathscr{P} - C_2\varepsilon) \quad . \end{aligned} \tag{4.53}$$

Details of the model can be found in 2.2.1.

Instead of solving this equation system on the Lagrangian markers as in Section 4.2, we solve it on the coarse grid which is also used for the local evaluation Section 4.4.2. The model can also be solved on the high-resolution surface mesh, this did not yield a significant difference in our experiments. The reason for this is that the variables k and ε are averaged properties, and spatially vary smoothly due to turbulent diffusion. The Eulerian approach has the advantage that it is easier to exclude the effects of buoyancy, as discussed below.

The primary interest here is to compute source terms for driving the model. The sources should capture the wall-induced turbulence, but exclude turbulence induced by buoyancy. If we were to directly use k for injecting turbulence we would include the effects of buoyancy twice: once from the k–ε model, and once from the vortex sheet model. In addition, a general turbulence model would not be able to capture the characteristic effects of buoyancy, such as the cloud billowing. We therefore need to guarantee orthogonality of the two

methods, by excluding the effects of buoyancy from (4.53), such that each model can focus on the type of turbulence it is most suitable for. With a strain-based production term that is commonly used for the k–ε model, this would however imply separating the wall induced turbulence from the total one. This is, to the best of our knowledge, not possible for a strain based production. There is, however, an alternative production term $\mathscr{P}_R$ based on rotation. Compared to the strain based measure, it is less accurate for free-stream generation but still captures buoyancy and wall induced turbulence very well. Assuming we have a measure for the current buoyancy-induced turbulence, we can subtract it from $\mathscr{P}_R$ to single out the turbulence induced by obstacles. We have found that using the rotation-based production term from Spalart [SA94] and a vorticity based integration of the buoyancy production allows us to do just this.

According to Spalart [SA94], the production is given by

$$\mathscr{P}_R = 2\nu_T \sum_{i,j} \Omega_{ij}^2 \tag{4.54}$$

with the rotation tensor Ω_{ij}. We now express its tensor norm in terms of vorticity as $\sum_{i,j} \Omega_{ij}^2 = |\boldsymbol{\omega}_f|^2$. Here $\boldsymbol{\omega}_f$ is simply the vorticity of the grid-based flow field given by $\boldsymbol{\omega}_f = \nabla \times \mathbf{U}$. With $\boldsymbol{\omega}_g$, which denotes the buoyancy induced vorticity strength that we will compute below, we obtain turbulence production for purely wall-generated turbulence using the difference of the two:

$$\mathscr{P}_{wall} = 2\nu_T |\nabla \times \mathbf{U} - \boldsymbol{\omega}_g|^2 \quad . \tag{4.55}$$

For stability, we ensure that $|\nabla \times \mathbf{U}| \geq |\boldsymbol{\omega}_g|$. An example from the simulation of Fig. 4.26 comparing the two vorticity measurements can be found in Fig. 4.32. Finally, we need to compute the accumulated vorticity induced by buoyancy $\boldsymbol{\omega}_g$. Applying the Boussinesq assumption and omitting external forces, we obtain an evolution equation for the buoyant vorticity $\boldsymbol{\omega}_g$ with

$$\frac{\mathrm{D}\boldsymbol{\omega}_g}{\mathrm{d}t} = \boldsymbol{\omega}_g \cdot \nabla \mathbf{u} + \frac{1}{\rho}(\nabla \rho \times \mathbf{g}) \quad . \tag{4.56}$$

We integrate this equation over time on the grid in combination with the k–ε model to obtain the wall based turbulence production $\mathscr{P}_{wall}$ as outlined in Fig. 4.31. Equipped with this production term we compute the spatial distribution of the turbulent kinetic energy k that we use to synthesize turbulent detail on the smoke surface.

Turbulence Synthesis

In contrast to buoyancy induced turbulence, we can synthesize the turbulence triggered by our obstacle-induced turbulence model using K41 theory Section 2.3. In this regime energy is mainly scattered from large to small scales, so we can approximate the velocity of the turbulent details using a frequency-matched curl noise texture that is advected through the large-scale velocities, as described in Section 2.4.1. Instead of evaluating the turbulence at each cell of a higher resolution grid, we can synthesize it more accurately on the mesh. Each mesh node carries a texture coordinate $\mathbf{q}$ for curl noise texture, and its turbulent kinetic energy k is interpolated from the grid. The additional velocity per node is then given by

$$\mathbf{u}_D(\mathbf{r}) = \nabla \times \sqrt{\alpha_S k}\, \mathbf{N}_\mathbf{f}(\mathbf{r}) \tag{4.57}$$

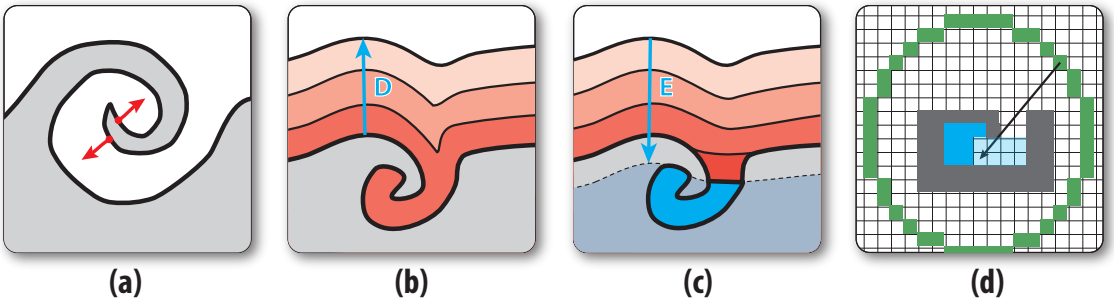


Figure 4.33: To simplify mesh geometry, we collapse invisible thin sheets. We fist identify candidate nodes in very thin sheets (a). Next, we compute an eroded inside volume on grid in steps (b) and (c). Finally, we check whether these cells are visible with a raycast towards an enclosing sphere (d). All thin sheet nodes in the blue region of (d) are marked for edge collapses.

where $\mathbf{N_f}$ are the curl noise functions and α_S is a scaling parameter to control turbulence strength. We will demonstrate the interplay of the two turbulence models and their orthogonality in Section 4.4.5.

4.4.4 Implementation

In this section, details and parameters of our implementation in respect to turbulence estimation, mesh resampling and rendering are specified.

Turbulence Model

To solve (4.53) on the grid, we perform operator splitting as for the Navier-Stokes equations. The advection of k and ε in the PDE system is treated identical to the velocity self-advection using the MacCormack algorithm. The diffusion component $\nabla(\frac{\nu_T}{\sigma_k}\nabla k)$ is expressed using finite differences, with substepping if the CFL condition is violated. To prevent instabilities in the k–ε model for low turbulence intensities we ensure that k and ε are always in a meaningful range where a minimal amount of ambient turbulence is present. Bounds for k are given in terms of turbulence intensity I as $k = \frac{3}{2}U_0{}^2I^2$, with the characteristic velocity U_0 which is an estimate of the velocity scale in the simulation. We use $I_{min} = 10^{-3}$, $I_{max} = 1$. We found ε is best limited using the equation for the turbulent viscosity ν_T, as this parameter linerly affects production. In our experiments, $\nu_{min} = 10^{-3}$, $\nu_{max} = 5$ are used. As starting parameters for a weakly turbulent initial state we found $\nu_T = 0.1$, $k = 0.1$ to produce stable results.

Mesh Resampling

Due to advection and buoyancy, the mesh will undergo strong deformations. On the other hand, Gaussian smoothing and buoyancy integration rely on a relatively uniform mesh geometry. Therefore, we split and collapse triangle edges to keep all edge lengths l in the range $\Delta l < l < 2\Delta l$, where Δl is the desired minimal edge length. Vortical forces smaller

this minimal length would only be visible as a slight noise on the surface. So we use the regularization parameter α_R in (4.50) to enforce a minimum vortex size larger than Δl. For our example scenes, we chose $\alpha_R = 2\Delta l$. Finally, we apply a small amount of explicit Laplacian smoothing to the mesh [DMSB99], to prevent the accumulation of small-scale noise on the surface.

The vortical motion on the mesh interface creates vortex roll-ups, which lead to the generation of spiral-shaped thin sheets. Since vorticity generation is linked to the surface normal, both sides accumulate almost equal amounts of vorticity, with opposing direction vectors. As the sheets become thinner, the vorticity effect on surrounding nodes therefore becomes smaller and effectively cancels out. Also, many of these thin structures are typically hidden inside the bulk volume of the cloud. On possiblity to reduce the complexity somewhat based on these two observations as presented in [PTG12] is explained below.

Thin sheets are identified, and the ones which are invisible from the outside are removed. First, we mark nodes on thin sheets, check which of these are far inside volume, and finally perform a visibility test to determine nodes not visible from the outside. The process is visualized in Fig. 4.33.

As a first step, *thin sheet nodes* are identified by checking for a vertex with opposing normal ($\pm$ 20°) within close proximity, i.e. at a distance less than Δl opposing the vertex normal. This can be done efficiently using the grid as acceleration data structure. Next, we identify the volume inside the cloud on the grid. As a coarse representation of the outer hull, we first compute a level set for the mesh. Since triangle size is always well below the size of a grid cell, we can employ a simple and fast method [Kol05] to obtain the signed distance function (SDF). We then enlarge and shrink the level set to close small holes and cavities induced by the complex mesh geometry. The level set is enlarged by $D = 4$ cells to compute an outer interface. We rebuild the SDF at a distance $E = -(D+2)$ from this interface, to obtain a faired volume slightly smaller than the original one. All cells inside this volume are marked as *inside cells*.

As cells in a cavity might still be visible from the outside, we finally compute visibility for the inside cells by performing a raycast towards target points on a sphere enclosing the surface mesh. The cost for these tests is less than 5% for our simulations, as there are typically few cells to be tested. All thin sheet nodes that are located in cells identified as not visible from the outside are marked to be collapsed during the next edge collapse step in line 25 of Fig. 4.31.

It has to be noted that while this algorithm is able to reduce mesh complexity, it is rather conservative and useful mainly in scenarios with a strong entrainment but coherent surface. Other methods such as [WTGT10, BKB12] could be employed to help to reduce complexity; however, there also introduce severe overhead for meshes of this complexity and may cost more than they bring in terms of performance benefit.

Rendering We use three different methods to render the simulation results.

- For very dense volumes, the mesh could be displayed directly. However, we have found that it is beneficial to add a certain amount of transparency for very thin structures. In the shader, we check the thickness of the volume. If it is above a certain threshold, we render it opaque, otherwise semi-transparently with a transparency proportional to its thickness. We use a threshold of $\Delta x/2$ in our examples.

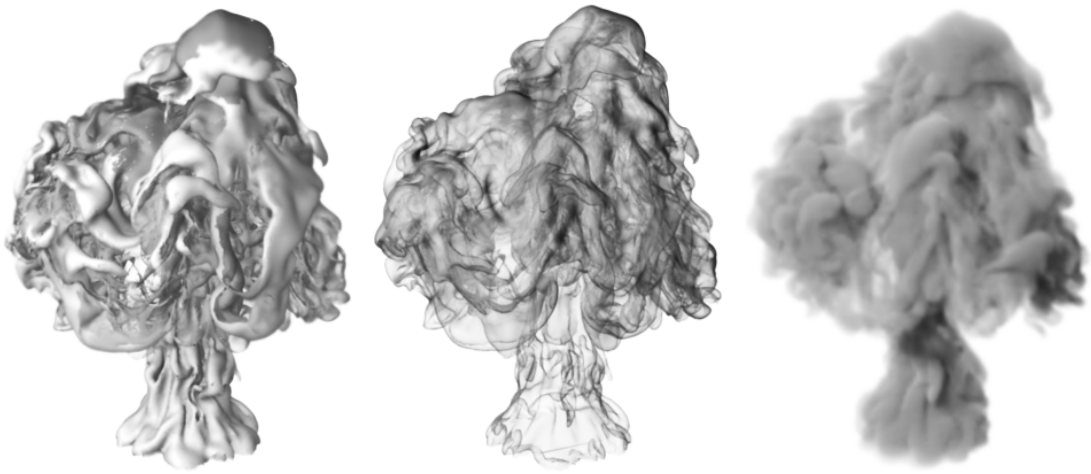


Figure 4.34: A plume is rendered using semi-transparent rendering (left), wispy smoke rendering (middle) and volume rendering (right). While volume rendering produces the most realistic results for dense plumes, semi-transparent and wispy rendering enhance the visualization of the vortex sheet structure.

- To emphasize the detailed structures from the surface vorticity model we can leverage the fact that smoke often concentrates on the vortex sheets [Sto06]. To highlight these surfaces, we modulate the transparency by an approximation term for smoke sheets as given in Funck et al. [vFWTS08]. To prevent the apparent increase of smoke density by elongation of the mesh, we track the smoke concentration at each triangle during simulation. It is seeded with a constant value at the inflow, and distributed during remeshing. This per-triangle concentration is multiplied onto the transparency during rendering.
- Lastly, it can be useful to leverage the commonly used volumetric shaders of an existing rendering pipeline. To do this, we project the mesh onto a grid data structure. This density grid might require a high resolution, but is independent of the simulation resolution and only required for rendering.

The effect of these different rendering techniques can be seen, e.g., in Fig. 4.34. For most of the example scenes we have used the semi-transparent shader, the only exception is Fig. 4.36, where we used the volumetric shader.

Performance For high-resolution triangle meshes, the two most costly steps in the simulation loop are applying the Gaussian kernel to the mesh, and integrating (4.52). However, these operations are simple and do not depend on neighborhood information. Therefore, they are very suitable for parallelization. Using GPU computing with CUDA, we obtained significant speedups of approximately a factor of 10. In the CUDA routine for calculating the velocity update, we use a precomputed hash grid structure to exclude triangles outside the cutoff radius. We note that the complexity can be further reduced using Treecodes, e.g. [QV01]. For our example scenes with a few hundred thousand vertices, we however found our simple approach to be sufficient.

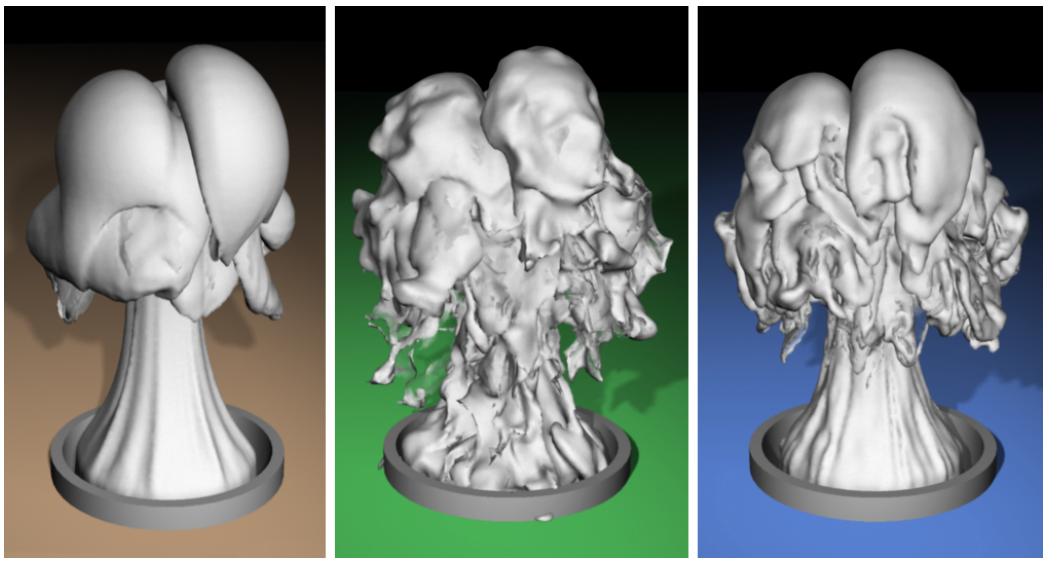


Figure 4.35: We compare the simulation of a buoyant plume with isotropic turbulence modeling (middle) to our method (right). The base simulation is shown on the left. While isotropic turbulence creates unrealistic surface distortions, the turbulence onset is calculated correctly using our approach.

4.4.5 Results

In the following, we demonstrate the properties of our model based on several simulations setups.

Turbulence onset To demonstrate the ability of our vortex sheet dynamics to correctly compute the turbulence onset, we simulated a buoyant smoke plume as shown in Fig. 4.35. The setup uses $64 \times 96 \times 64$ grid cells for the base solver, and a triangle edge length $\Delta l = 0.18\Delta x$. Without artificial disturbing forces, the base flow remains smooth and does not show any turbulent detail. To demonstrate the effect of standard turbulence methods, we synthesize turbulence using vortex particles. The vortex particles are emitted at the inflow and moved along the flow with the smoke plume. For the particles, we use a size and energy distribution based on the Kolmogorov spectrum. This is typically a good assumption for bulk volume flows, as isotropization drives the turbulence towards a Kolmogorov spectrum eventually. At the interface, however, the length scales are model-dependent and production is highly anisotropic. This leads to a lack of coherent features using isotropic turbulence methods. Using our method, we observe that the generated detail organically integrates with the large-scale flow.

Eulerian-Lagrangian coupling We demonstrate the generality of our model by simulating two setups with more complex boundary conditions. The first scene, depicted in Fig. 4.36, shows strongly billowing clouds moving through a channel of irregularly shaped obstacles. We simulate an expanding front of smoke with density slightly above air, with a base resolution of $40 \times 40 \times 128$. It can be seen that the flow easily follows the geome-

Figure 4.36: In this example scene, an expanding, turbulent smoke front is simulated. The typical cloud billowing is clearly visible in the smoke plume shape. This effect can not be achieved using turbulence synthesis.

try of the scene due to the Eulerian simulation, while our vortex sheet model leads to the development of the typical billowing cloud surfaces. In the second scene, the interaction between water and a heavier liquid is simulated. We use a base solver with $96 \times 64 \times 64$ grid cells, and pulsed inflow conditions to simulate the injection of multiple drops of fluid. In this case, the temporally changing inflow leads to complex density surfaces developing over time from the buoyant turbulence. Note that the irregular walls of the first, and the pulsed inflow of the second example would be difficult to realize with a simulation based on a pure vorticity formulation.

Wall turbulence In a next example, the interplay between mesh buoyancy and our turbulence model is investigated. To this end, we simulate a plume under the influence of buoyancy and a moving obstacle. Fig. 4.26 shows the orthogonality of the both models: with only the turbulence model activated, we observe detailed structures forming in the wake of the obstacle, while the rest of the flow remains laminar. Once the vortex sheet model is enabled, the mesh shows small-scale deformations with correct orientation due to buoyancy. We show that by combining the two models, we can benefit from both the accurate prediction of source regions by the turbulent energy model, as well as the anisotropic generation of the vortex sheet method. This example exhibits a large number of highly de-

Setup	Grid res.	#tris mio.	$\Delta l/\Delta x$	Mesh [s]	Grid [s]
Bunny Fig.4.26	$64 \times 64 \times 64$	0.9 / 2.6	0.2	9 / 33	0.6
Water Fig.4.30	$96 \times 64 \times 64$	0.8 / 3.2	0.15	12 / 40	1.3
Plume Fig.4.35	$64 \times 96 \times 64$	0.6 / 2.3	0.18	7 / 22	0.6
- w/o cutoff	$64 \times 96 \times 64$	0.6 / 2.4	0.18	36 / 101	0.5
- base only	$64 \times 96 \times 64$	0.2 / 0.8	0.18	1 / 6	0.5
- vortex part.	$64 \times 96 \times 64$	0.4 / 1.5	0.18	5 / 16	0.6
Street Fig.4.36	$40 \times 40 \times 128$	1.0 / 1.8	0.2	11 / 41	0.9
Duck Fig.4.37	$64 \times 96 \times 64$	0.8 / 3.1	0.2	8 / 30	0.4
- VIC 64	$64 \times 96 \times 64$	0.1 / 0.3	0.2	0.2 / 0.4	6 / 16
- VIC 256	$256 \times 384 \times 256$	0.8 / 3.8	”	4 / 11	156 / 350

Table 4.3: Performance measurements for our simulation runs. Timings are mean runtime per frame. Two values with a ”/” denote the mean and maximum values, respectively. *Grid* refers to all Eulerian operations, while *Mesh* represents vortex sheet dynamics. All simulations were run on a workstation with an Intel Core i7 CPU, a NVidia GTX 580 graphics card and 8GB of RAM.

tailed swirls, many of them less than a fifth of a cell in diameter. These surface details are not smeared out despite moving along with the fast and turbulent velocities. Representing this detail during the course of a purely grid-based simulation would require a large amounts of memory, and corresponding amounts of computation for the advection step.

Performance The two most costly steps are applying the Gaussian kernel to the mesh, and integrating (4.52). Since these operations are simple and do not depend on neighborhood information, we evaluate them on the GPU. This leads to an average time of 10s per frame for the example scenes shown. The majority of this time is spent on the vortex sheet evaluation, i.e. the performance primarily depends on the number of triangles in the mesh. The number of triangles is in turn determined by two factors: the shot length, as triangle numbers typically increase during the course of a simulation, and the re-meshing resolution Δl. The parameter Δl can therefore be used as a means for fine-tuning detail versus performance. The performance numbers and statistics for all scenes can be found in Table 4.3, where *base only* refers to the plume simulation without a turbulence model.

In Fig. 4.37 it is demonstrated that the method can obtain the same quality of results as the Vortex-in-Cell (VIC) scheme used, e.g., in Stock et al. [SDT08]. It is howver considerably faster than VIC and interfaces better with e.g. inflows and turbulence models due to its hybrid nature.

4.4.6 Conclusion

In this chapter, an algorithm for simulating buoyant, turbulent smoke plumes was presented. A Lagrangian surface mesh is used to track the smoke/air interface. On this mesh, we solve the vortex sheet dynamics, and couple it to a low-resolution Eulerian fluid solver. This allows correctly simulating the turbulence generation process on the interface, which is important for visual coherency. On the other hand, the coupling with Eulerian large-scale

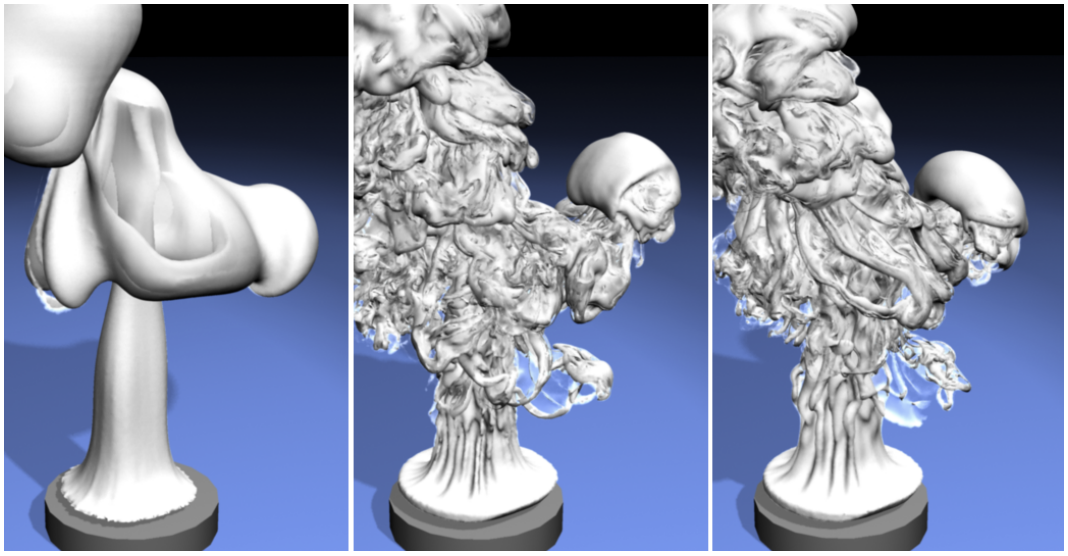


Figure 4.37: We compare our method to Vortex-in-Cell integration. Our approach (middle) produces similar results as VIC on a 256 grid (right), while being 19 times faster. On the other hand, VIC with a resolution of 64 (left) has a comparable runtime to our method, but exhibits significantly less detail.

dynamics allows evaluating the update of the velocity in a purely local fashion. This greatly reduces the complexity, and enables the efficient simulation of detailed plumes with non-trivial static boundaries or moving obstacles. In addition, we have proposed an orthogonal turbulence model for capturing turbulence production from obstacles.

A limitation of this approach is that it can lead to meshes with large numbers of triangles. Due to re-meshing, the number of triangles will increase over time in turbulent regions for long simulation times. Also, accumulated integration errors and re-meshing operations can lead to self-intersecting surfaces. Although our resampling approach may reduce the complexity of the meshes, it can not keep up with the growth in complexity for long shots. For this, more aggressive approaches are needed. Other methods such as [WTGT10, BKB12] could be employed to help to reduce complexity; however, there also introduce severe overhead for meshes of this complexity and may cost more than they bring in terms of performance benefit. Investigating better resampling schemes might be an interesting topic for future work.

This method is naturally not well-suited for diffuse, hazy smoke. It would however be very interesting to combine our approach with a lower-resolution volumetric density representation. Sharp, detailed interfaces could then be tracked with our method, while the developing diffuse haze around the dense cloud could be represented on the volumetric grid. It would also be possible to add further detail based on the texture coordinates of the mesh, as we have a temporally coherent discretization of the surface over time.

4.5 Conclusions

4.6 Application Guidelines

The new techniques introduced in this thesis together with previous methods summarized in Section 2 are best seen as a toolbox of methods for simulating complex, turbulent flows. Many of the elements can be interchanged, or combined in different ways depending on the requirements. This section provides guidelines for the practical use of these individual components. Most of them are also available as modules in the open source fluid solver *Mantaflow*, which can serve as a framework for experimentation and research in turbulent fluid dynamics.

Turbulence Prediction The simplest turbulence predictors are vorticity [FSJ01] and wavelet-decomposition of the velocity field [KTJG08] (Section 4.1.4). These prediction are very easy to implement, and the first thing to try. To simply add some turbulence to an existing simulation, especially the later is sufficient and proven. Strictly speaking, however, these prediction only produces meaningful results for strongly forced turbulence, and will fail in most complex cases, especially when using low resolution base solvers. Even the standard test case of a rising plume does not fall in this regime, as it strongly relies on interface dynamics, and will look incorrect on closer inspection. The arguably most useful representation for non-trivial turbulence prediction is TKE, as a vast set of well-proven tools exists for this representation by the means of classical turbulence models. For many use cases, a turbulence predictor based on a complete two-equation model such as the k–ε model Section 4.2.2 provides the best trade-off between complexity and prediction power. On the one hand, the simpler incomplete one-equation models require scene-dependent information such as a mixing length, which are hard to specify in the general case. On the other hand, more complex models such as full Reynolds stress transport rarely pay off for Graphics applications. While they provide more prediction power especially for highly anisotropic and transition flows, it is hard to use the information gained in a meaningful way, as accuracy is limited by the statistical synthesis methods.

Turbulence Synthesis The most popular turbulence synthesis method in Graphics is frequency-matched curl noise texture synthesis as described in Section 4.1.4. This is due to their simplicity, efficiency and the fact that they work well in combination with TKE predictors. Instead of representing and simulating turbulence dynamics, only a texture lookup has to be performed, which makes it the prime choice in methods geared towards real-time such as Section 4.2. However, this method suffers from a number of severe drawbacks. Firstly, the transition between coherent anisotropic structures and the isotropic textures creates visual artifacts. This can be partly alleviated by 2D anisotropy extensions as described in Section 4.2, but the method is inherently limited in that detail structures cannot easily be edited or aligned to coherent flow features. Therefore, it will always remain disconnected from the base flow. Also, the modulation of the noise texture with the TKE effectively creates divergences, which may be a problem if strong gradients of turbulence intensity exist in the scene. Even more importantly, the detail dynamics is limited by the static nature of

texture. Within an octave, there is no interaction between the generated turbulent eddies, which creates an unrealistic frayed-out look especially if no background flow is present.

For small synthesis scales, i.e. using a high resolution base solver to cover the mid-range turbulence, for flows with mostly homogeneous turbulence intensities or for real-time scenarios, curl noise texture synthesis is therefore a good choice. For all other cases it pays off to directly represent turbulence using a vortex representation, such as vortex particles (Section 4.3), filaments [WP10] or vortex sheets (Section 4.4). This most often results in more plausible turbulence dynamics, and allows to model more complex turbulent effects such as transition and breakdown. On the downside, it takes more effort to couple these representations to turbulence predictors, and re-meshing can be an issue. To represent strong turbulence, vortex particles are the prime choice, as these flows tend to be less connected and represented most compactly using particle kernels. Filaments are very efficient to cover mid-level turbulence, and are also useful for the modeling of transition effects. For interface effects, vortex sheets are most efficient. They are also the suited best for baroclinity-driven effects, such as cloud billowing which is hard to model in other representations.

Chapter 5

Liquid Turbulence

Turbulence on a liquid surface is a phenomenon that is distinct from the phenomena that we have seen previously. While velocity variations in the surrounding air and water play a role in the creating the detailed waves and wrinkles on a liquid surface, they do not tell the whole story. It is appealing to think that the ripples on the surface of a liquid are mere images of the vortices present in the surrounding fluid, but laboratory measurements in the physics literature [SvdW08] have shown that this appealing picture predominantly applies to the low frequency components of the liquid surface. As the frequency increases, new dynamics emerge that are distinct from the velocities in the surrounding fluids.

If the surface variations on a liquid surface were directly enslaved to the turbulence of the underlying velocity fields, we would expect that some statistic of the surface, e.g. the surface gradient, would follow the same power law as the velocity field. In essence, we would expect the $-\frac{5}{3}$ Kolmogorov spectrum to appear somewhere in relation to the surface. However, wave turbulence theory [ZLF92] predicts a much steeper exponent, $-\frac{11}{4}$, otherwise known as the *Kolomogorov-Zakharov* (KZ) spectrum. The larger negative exponent implies that high frequency surface waves are much less persistent than high frequency velocities. This general implication is supported by laboratory experiments, but the effect has been observed to be even more extreme, as even larger exponents, e.g. -6, have been measured [SvdW08]. Other works [Fal10] have also reported measurements that are not in line with the theory. These highlight the fact that, relative to single phase turbulence, free surface turbulence is a phenomenon that still contains many more unanswered questions. Given the distinct physical nature of this phenomenon, it is usually referred to as "wave turbulence" or "weak turbulence". Many excellent survey papers are available [BP01, DK99] on the topic.

Fortunately, there is numerical evidence that the high frequency components on liquid surfaces can be approximated using advected wave sources ([Sav06], Chapter 7.3). In light of this, it should be possible to add novel detail to the surface of an existing liquid simulation by running a couple wave simulation along its surface. Many wave models exist in addition to the classic wave equation, and we will begin by briefly describing several of them.

5.1 Wave Models

5.1.1 The Classic Wave Equation

The classic wave equation can be stated as,

$$\frac{\partial^2 h}{\partial t^2} = c\nabla^2 h \tag{5.1}$$

where h denotes the height of a 2D liquid surface. In computer graphics, this is a very popular model for wave motion that lends itself to straightforward integration schemes and appears in many works, e.g. [TWGT10a, ATBG08, KM90].

5.1.2 The Korteweg-de Vries Equation

A very well-studied model for wave motion is the Korteweg-de Vries (KdV) equation [Joh97, Tre00], which contains both a quadratic non-linearity and a third spatial derivative that enables dispersion:

$$\frac{\partial h}{\partial t} + h\frac{\partial h}{\partial x} + \frac{\partial^3 h}{\partial x^3} = 0. \tag{5.2}$$

The KdV equation is only one-dimensional, and its two-dimensional generalization is the Kadomtsev-Petviashvili (KP) equation:

$$\frac{\partial}{\partial x}\left(\frac{\partial h}{\partial t} + h\frac{\partial h}{\partial x} + \frac{\partial^3 h}{\partial x^3}\right) + \frac{\partial^2 h}{\partial y^2} = 0. \tag{5.3}$$

It is immediately apparent that KP equation is anisotropic, as the treatment of spatial derivatives is not symmetric. The asymmetry arises because it is assumed that the wave in the y direction has relatively low frequency. For this reason, we have shied away from using the KdV and KP equations. However, we note that the direction of principal variation could certainly be detected in a liquid simulation, allowing the KP equation to be oriented and applied. There is a wealth of powerful techniques related to the *inverse scattering transform* that could then potentially be brought to bear on the problem [DJ89].

5.1.3 The Non-Linear Schrödinger Equation

The KdV equation applies specifically to shallow water (long wave) scenarios. A more general, and also well-studied, alternative is the non-linear Schrödinger (NLS) equation, which takes the form:

$$i\frac{\partial h}{\partial t} + \alpha\frac{\partial h}{\partial x} = \beta|h|^2 h, \tag{5.4}$$

where α and β are constants, $i = \sqrt{-1}$, and the real component of the solution is the one of interest. In the shallow water limit, the NLS equation is known to reduce to the KdV equation. Like the KdV equation, the NLS equation is one-dimensional, so a two dimensional generalization is needed. For the case of water waves, the 2D analog to the NLS equation

is the Davey-Stewartson system [DS74]:

$$i\frac{\partial h}{\partial t} + \lambda\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} - \nu|h|^2 h = h\frac{\partial \phi}{\partial x} \tag{5.5}$$

$$\alpha\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = \xi\frac{\partial |h|^2}{\partial x}, \tag{5.6}$$

where ϕ denotes the velocity potential. Much additional investigation has been performed on these equations, including the addition of surface tension [DR77] and a deep water generalization [Hog85]. By looking at the right hand side of these equations, it is clear that they still display anisotropies similar to the KP equations, so we again preferred not to use them, but again note that inverse scattering transform methods could potentially be applied.

5.2 The iWave Algorithm

We elected to use the iWave algorithm [Tes04b, Tes04a], because it is known to introduce more visual variety to wave simulations, is used extensively in production [Car07, FH09, AAB+11], and inherits much of the simplicity of the classic wave equation. One of the visual features that the more sophisticated non-linear models described above enable is *dispersion*, where a coherent feature gradually breaks apart into features with different frequencies. As we will see, the iWave algorithm also supports some form of this behavior. We note that the NLS and KdV equations support additional non-linear effects, such as phase shifting when two wave fronts collide. As the iWave algorithm is linear, it does not support this behavior. Finally, we found was that the custom-built kernel at the heart of iWave has larger spatial support than the Laplacian from the classic wave equation, which has the positive side off of making the overall integrator significantly stabler.

5.2.1 The iWave Equation

The central iWave equation closely resembles the classic wave equation:

$$\frac{\partial^2 h}{\partial t^2} = c\sqrt{-\nabla^2}h, \tag{5.7}$$

but instead employs the less conventional fractional Laplacian $\sqrt{-\nabla^2}$ in lieu of the traditional ∇^2. Following the reasoning in Tessendorf [Tes04b], the radical arises as follows. Assume that we have a potential, ϕ, and that the fluid velocity $\mathbf{u}$ can be retrieved by taking its gradient, $\mathbf{u} = \nabla\phi$. For a wave simulation, we are only concerned with the vertical component, h, so if we assume it points in the y direction, the term of interest is:

$$\frac{\partial h}{\partial t} = \frac{\partial \phi}{\partial y}. \tag{5.8}$$

Mass conservation must also be enforced in the liquid, $\nabla^2\phi = 0$. Separating this equation out into its constituent terms we obtain:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\phi = 0. \tag{5.9}$$

The temptation is to discard the $\frac{\partial^2}{\partial y^2}$ term, as there is no significant variation in the vertical direction. Doing this would in fact yield the classic wave equation. If we instead collapse the other two derivatives into the term $\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}\right) = \nabla_\perp^2$, we can obtain:

$$\begin{aligned}\left(\nabla_\perp^2 + \frac{\partial^2}{\partial y^2}\right)\phi &= 0 \\ -\nabla_\perp^2 \phi &= \frac{\partial^2 \phi}{\partial y^2}.\end{aligned}$$

The radical from Eqn. 5.7 then appears because we can take the square root of both sides:

$$\begin{aligned}\sqrt{-\nabla_\perp^2 \phi} &= \sqrt{\frac{\partial^2 \phi}{\partial y^2}} \\ \sqrt{-\nabla_\perp^2}\,\phi &= \frac{\partial \phi}{\partial y}.\end{aligned}$$

We can now plug this is into our original definition of velocity potential, Eqn. 5.8:

$$\sqrt{-\nabla_\perp^2}\,\phi = \frac{\partial h}{\partial t}. \tag{5.10}$$

This almost matches Eqn. 5.7, except that the ϕ term remains. If we could instead obtain an equation that only contains h, integrating it forward in time would be much easier. In order to do this, we apply a time derivative to both sides:

$$\sqrt{-\nabla_\perp^2}\,\frac{\partial \phi}{\partial t} = \frac{\partial^2 h}{\partial t^2}, \tag{5.11}$$

and the perform a substitution based on the linearized Bernoulli equation,

$$\frac{\partial \phi}{\partial t} = ch, \tag{5.12}$$

to obtain the final iWave equation:

$$c\sqrt{-\nabla^2}\,h = \frac{\partial^2 h}{\partial t^2}. \tag{5.13}$$

5.2.2 Spatial Discretization

Having established Eqn. 5.7, the next step is to spatially discretize the fractional Laplacian, $\sqrt{-\nabla^2}$. Tessendorf [Tes08] converts the operator to a Fourier representation to obtain the following convolution kernel:

$$G(\mathbf{x}) = \int_0^\infty k^2\, e^{-k^2\sigma^2} J_0(k|\mathbf{x}|)\, dk, \tag{5.14}$$

where k is frequency, σ is a Gaussian cutoff parameter, the J_0 is the zeroth Bessel function of the first kind. This equation can be difficult to parse, so we will try to impart some intuition here.

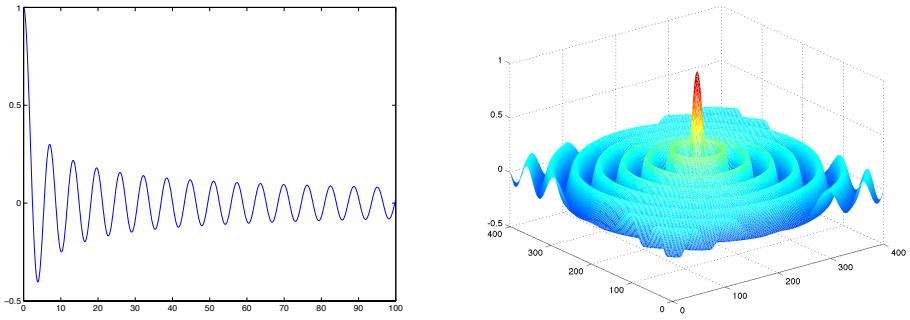


Figure 5.1: **Left:** The J_0 Bessel function. **Right:** The $J_0(|\mathbf{x}|)$ function plotted using a 2D coordinate $\mathbf{x}$. It is essentially the top plot, swept in a circle.

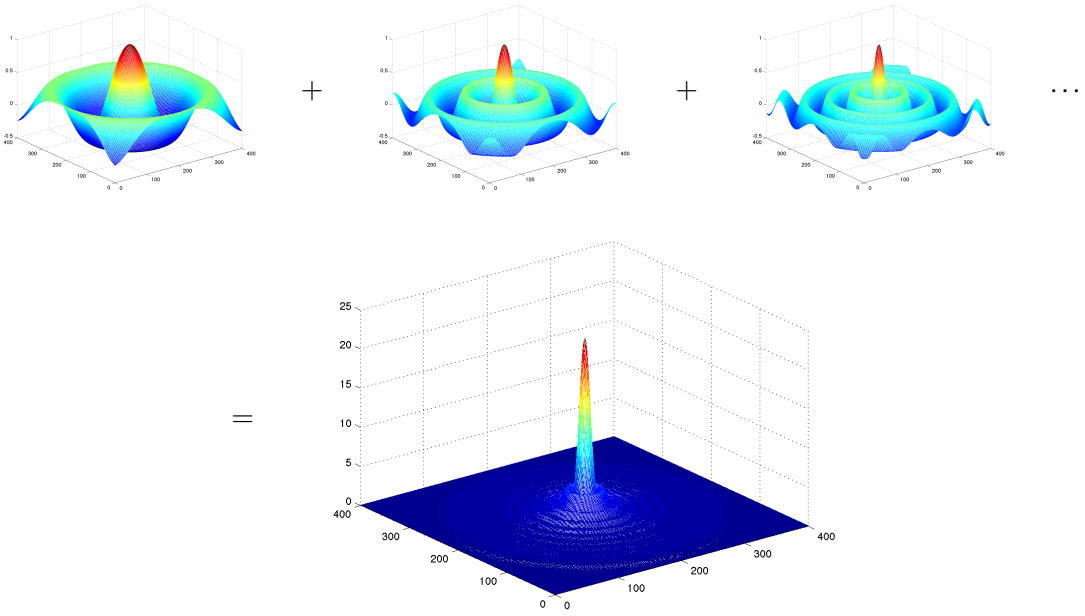


Figure 5.2: The integral $\int_0^\infty J_0(k|\mathbf{x}|)\,dk$ is a summation of 2D Bessel functions over a set of frequencies.

We will first look at solely the $J_0(\mathbf{x})$ term. The J_0 Bessel function is an oscillatory function that decays fairly slowly in space, as can be seen on the left of Fig. 5.1. When $\mathbf{x}$ denotes a 2D spatial coordinate and $J_0(|\mathbf{x}|)$ is evaluated in the 2D plane, we obtain the wavy shape seen on the right of Figure 5.1.

The integral in Equation 5.14 denotes a summation of many of these 2D functions with increasing frequency, as can be seen in Figure 5.2. Notably, the summed functions tend to cancel each other, so the integral falls off faster in space than its constituent Bessel functions. However, adding these functions directly does not generate a fractional Laplacian kernel, as

the spectrum obtained is not correct. Each Bessel function should instead be weighted by the square of its frequency, $\int_0^\infty k^2 J_0(k|\mathbf{x}|)\, dk$, as can be seen in Fig. 5.3.

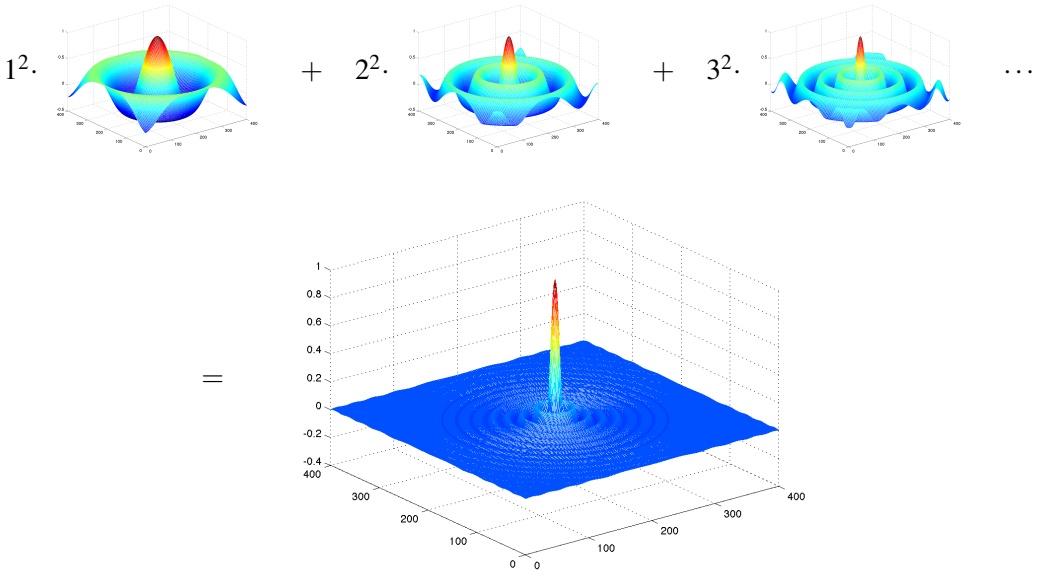


Figure 5.3: The integral $\int_0^\infty k^2 J_0(k|\mathbf{x}|)\, dk$ weights each Bessel function according to the weights dictated by the fractional Laplacian.

Tessendorf observes that this integral is actually divergent; as the frequency k grows larger, the weight k^2 grows quadratically, so in the limit $k \to \infty$, the integral sums to ∞. In order to address this problem, he adds a "soft cutoff" Gaussian term, $e^{-k^2\sigma^2}$, which shrinks faster than the k^2 term, and thus suppresses the divergence. Note that this Gaussian is in the frequency domain, not the spatial domain. By combining the three components we have just described, J_0, k^2, and $e^{-k^2\sigma^2}$, we obtain the full term under the integral in Eqn. 5.14. The final resulting kernel can be seen in Fig. 5.4.

5.2.3 Time Discretization

With the spatial discretization of the fractional Laplacian in place, we can now turn to devising a temporal discretization. Again following Tessendorf [Tes04b] we add a damping term to the iWave equation in order to maintain stability:

$$\frac{\partial^2 h}{\partial t^2} + \alpha \frac{\partial h}{\partial t} = c\sqrt{-\nabla^2} h, \tag{5.15}$$

where α denotes a damping constant. If we denote our iWave convolution kernel as G, we can now insert simple finite difference stencils into the time derivatives:

$$\frac{h^{t+1} - 2h^t + h^{t-1}}{\Delta t^2} + \frac{h^{t+1} - h^t}{\Delta t} = Gh^t, \tag{5.16}$$

and obtain an update rule for h:

$$h^{t+1} = \frac{(2-\alpha\Delta t)h^t - h^{t-1} + \Delta t^2 G h^t}{1+\alpha\Delta t}. \tag{5.17}$$

5.2.4 A Preview of the 3D iWave Kernel

In subsequent sections (§5.5.3), we will be constructing a 3D version of the iWave kernel in order to synthesize waves on a liquid surface. The intuition behind the 2D kernel transfers fairly directly to the 3D domain. The J_0 Bessel function will be replaced with the $\text{sinc}(x) = \frac{\sin x}{x}$ function, but the role of the integral summing over various frequencies will remain the same (Figure 5.5).

5.3 Closest Point Turbulence

Finally, we propose two turbulence coupling methods that seed the high resolution wave simulation in visually expected regions.

Despite much recent progress, it remains a challenge to simulate large-scale, high-resolution liquids. The recently popular "up-res" approach separates simulations into large- and small-scale details, and runs separate algorithms for each scale. In addition to being more efficient, imposing a one-way coupling between scales can facilitate design. A user can interact with a fast, low-resolution simulation, and later add additional high-resolution detail in a way that does not invalidate the low-resolution design. This approach has been successfully applied to several natural phenomena, including cloth simulations [BMWG07] and single phase smoke simulations [KTJG08, NSCL08, SB08a, NCZ+09,

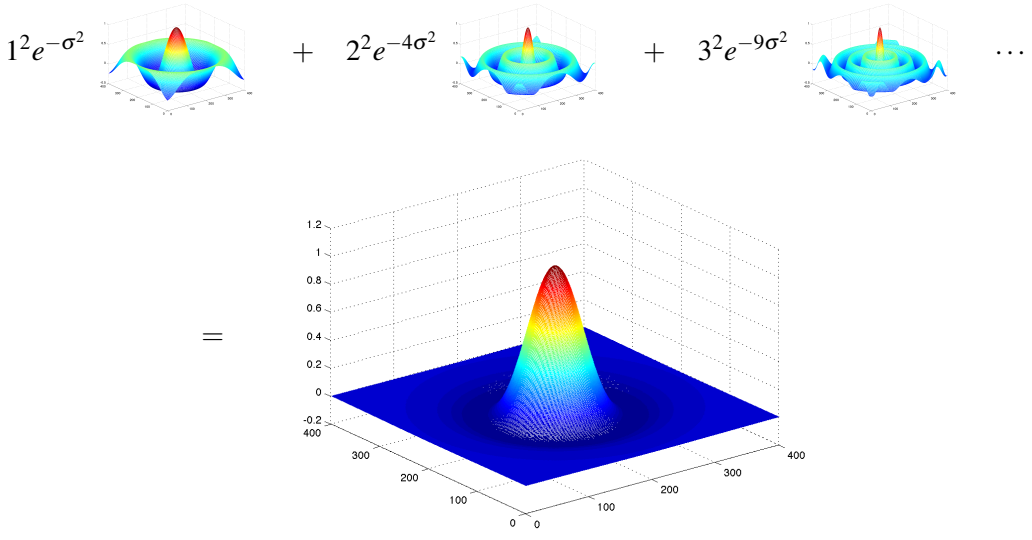


Figure 5.4: The final integral $\int_0^\infty k^2\, e^{-k^2\sigma^2}\, J_0(k|\mathbf{x}|)\, dk$ adds a "soft cutoff" term to avoid blowup at large k.

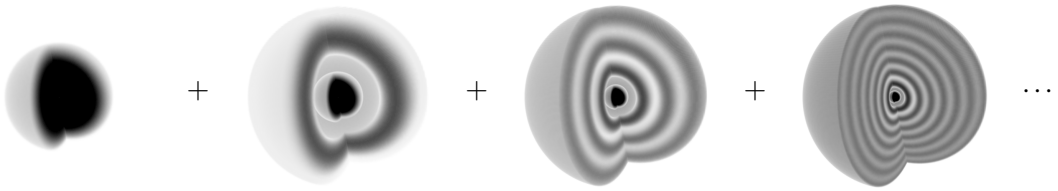


Figure 5.5: In 3D, the J_0 in the iWave kernel will become a sinc function, shown above with a quadrant removed to show the internal structure. The notion of scaling and summing over various frequencies to obtain the final kernel remains exactly the same.

HMK11, YCZ11]. However, it has been less successful for liquid simulation. Practitioners have reported [Lai11] that applying single-phase techniques to liquid simulations introduces undesirable artifacts and does not create plausible new details.

We posit that the reason for this is that the physics being simulated has been incomplete. Previous methods for increasing the resolution of liquid simulations [NSCL08, JKB$^+$10, YZC12] have assumed that if the turbulence of the underlying fluid velocity field is increased, high resolution surface dynamics will follow. However, the literature on free surface turbulence, also known as "wave turbulence" or "weak turbulence", maintains that the free surface, especially at high frequencies, possesses additional dynamics that are not mere images of the underlying velocity field. While the low frequency components of the velocity field initiate surface waves, many high frequency details arise from the independent oscillation of the surface membrane [SvdW08, Fal10].

We present a method that captures these additional dynamics by explicitly performing a wave simulation on the liquid surface. In doing so, we reduce the volumetric problem to a surface-only problem. We use the state-of-the-art in visual wave simulation, the *iWave* algorithm [Tes04b, Tes04a]. As we are simulating a scalar on a surface of rapidly changing topology, we immediately encounter the problem of consistently parameterizing a deforming surface. We sidestep this problem entirely by using a newly developed embedding method known as the *Closest Point Method* (CPM) [RM08]. The CPM operates on a 3D extension field instead of a 2D surface field, and thus requires no surface parameterization. However, it requires the existence of 3D spatial operators. Natural 3D analogs of 2D surface operators are often available, such as the 5-point 2D and 7-point 3D Laplacians. However, for many operators, such as the fractional Laplacian in the iWave algorithm, no obvious 3D equivalent is available, and it is unclear if the CPM can be used. We show that a viable CPM operator can be constructed by taking the *inverse Abel transform* of the original surface operator.

The CPM has predominantly been used on rigid 3D objects, where the cost of computing a closest point transform [Mau03] can be amortized. We instead deal with a deforming surface where the transform is computed and advected every frame. In order to prevent this from becoming the bottleneck, we propose an iterative transform based on the *Nacelle* algorithm [Tes11] that is faster than fast marching-based methods [AS03] and more effective at maintaining sharp features. Lastly, we propose two turbulence seeding methods that provide visually consistent methods of coupling the high-resolution surface simulation to the low-resolution volume simulation.

Our specific contributions are as follows:

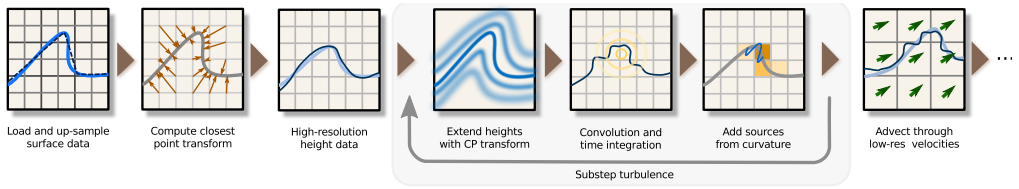


Figure 5.6: An overview of the different steps of our simulation algorithm. We read in data from an existing level set solver and add additional surface detail by performing a surface-only wave simulation. The final result can be used as a bump or displacement map during rendering.

- A method of constructing operators for the Closest Point Method when no natural 3D operator is available.
- A fast, iterative *Nacelle* method for building the closest point transform of a deforming surface.
- A *frozen core* version of the Nacelle method and an efficient narrow-band advection method that improves surface details.
- Two turbulence seeding strategies that introduce waves in visually expected regions.

5.4 Previous Work

Prior graphics work: Level set simulation of liquids was pioneered by Foster and Fedkiw [FF01]. Since then, many techniques have been proposed for simulating liquids with higher spatial resolution. Recent works include coarse grid projections [LZF10], higher order reinitialization methods [HK10], complementary Lagrangian meshes [WTGT09], and fast tall cell methods [CM11]. Bargteil et al. [BGOS06b] developed a method for texturing such simulations and successfully ran a reaction-diffusion simulation on the surface. However, significant surface parameterization problems arose, which led to subsequent work [BSM$^+$06, KAK$^+$07, NKL$^+$07] where the parameterization was synthesized each step. A surface texture was then synthesized from exemplars; no simulation took place. Our method sidesteps the parameterization problem entirely and allows a non-trivial iWave simulation to be performed on the surface.

One of the goals of our algorithm is to facilitate the design of liquid animations, so it can also be considered a liquid control algorithm. Many approaches, such as keyframes [MTPS04, SY05] and guiding shapes [NB11] have been developed to address this problem. Our method can be used to add additional surface detail to the results of these algorithms, so we consider them to be complementary.

A good survey of techniques for simulating ocean waves is available in Darles et al. [DCGG11]. While these techniques give good results for scenes without interaction, we do a full 3D simulation that automatically adds sources and handles obstacle interactions. Other recent work on wave simulation has included the development of fast, Lagrangian "wave particles," [YHK07], and the addition of the FFT algorithm described by Tessendorf [Tes04b] to a shallow water solver [CM10]. Several previous authors have attempted to simulate waves on deforming surfaces. Angst et al. [ATBG08] simulated waves on a fixed character mesh

surface that does not undergo any topology changes. They only simulated the traditional wave equation, not the iWave equation. Thürey et al. [TWGT10a] also simulated the wave equation on a Lagrangian mesh in order to capture surface tension effects. The focus of their method was on sheet breakup and large scale instabilities, so they did not achieve the fine-scale wave detail that we are able to produce. Kim et al. [KSK09b] simulated a vortex sheet along the liquid surface to capture high resolution interface effects. Again, their approach initiated detailed sheet breakup, which is orthogonal to the surface detail that we capture in this current work, and could be combined with ours to achieve highly detailed liquids.

The closest work to ours is Patel et al. [PTM09], which performs an orthogonal projection of a 2D iWave simulation onto a 3D river. This approach works best when the 3D liquid is well-approximated by a 2D plane, which is a well-founded assumption for rivers, but clearly not true for general liquids. Figure 5.9, for example, would be difficult to capture without introducing significant distortions, but is a trivial test case for our method. The algorithm also requires the user to manually specify turbulence injection sites, whereas we propose a method that injects turbulence automatically.

We use the *Closest Point Method* (CPM) [RM08], a level set-based, parameterization-free surface simulation method, to perform our iWave simulation. The CPM is not the first level set-based method proposed for simulating surface phenomena (see e.g. previous variational formulations [BCOS01, Gre06]), but it sidesteps many of the complexities present in previous methods, so we prefer it here. All of the level set-based methods require 3D generalizations of 2D surface operators, so even if a variational method was employed, the 3D iWave kernel we present in §5.5.3 would be needed. Other works have used the CPM to simulate fire [HZQW10], the wave equation, and the Navier-Stokes equations [AMT$^+$12]. All of these works deal with cases where the surface operators have obvious 3D analogs, such as the gradient and Laplacian. To our knowledge, ours is the first method that successfully uses an operator that is a non-trivial 3D generalization. Hong et al. [HZQW10] apply the CPM to deforming surfaces by propagating scalars between frames using the extension field. We present a fast, iterative method of computing the extension that could be used to accelerate their method, and detail-preserving mechanisms that could further improve their results.

5.5 A Free Surface Turbulence Algorithm

In this section, we describe our algorithm for simulating turbulence on a free surface. We still start with preliminaries on the Closest Point Method (CPM) and iWave algorithms, show how they can be unified, and then present the complete algorithm. A high level overview of our approach can be seen in Figure 5.6.

5.5.1 The Closest Point Method

The Closest Point Method [RM08] is an embedding method for simulating partial differential equations (PDEs) on arbitrary surfaces. As with other embedding methods, it works directly on 3D volumes that avoid the problems of traditional surface-based simulation, such as the construction of low-distortion surface parameterizations, and the de-

velopment of specialized surface-based operators such as the Laplace-Beltrami operator [WMKG07, CLB+09]. While the algorithm operates in 3D, it supports narrow banding, which allows it to scale according to the complexity of the surface, not the volume.

Similar to previous embedding methods, the CPM operates on the 3D *extension field* of the surface, which is constructed by assigning each grid point the scalar value of the nearest surface point. Simulation proceeds by applying the 3D version of the desired PDE to the extension field. For example, in the case of surface diffusion, the familiar 7-point Laplacian stencil would be used instead of a Laplace-Beltrami operator. More concretely, an explicit CPM for diffusing a surface scalar u_{2D} through T timesteps of size Δt on a fixed surface mesh would proceed as shown in Algorithm 1.

Algorithm 1: diffuseUsingCPM(u_{2D})

```
1 begin
2     Build the closest point transform, CP, of u_2D
3     Build the extension field, u_3D^1 = CP(u_2D)
4     for t = 1 to T do
5         u_3D^* = u_3D^t + Δt · ∇²u_3D^t
6         u_3D^{t+1} = CP(u_3D^*)
```

In this algorithm, ∇^2 corresponds to the 7-point Laplacian, and CP interpolates and propagates the scalar values at grid points adjacent to surface out to the entire volume. The algorithm is very similar to a basic 3D explicit integration, with the key addition of the extension step on Line 6. Despite its apparent simplicity, the CPM has been shown to produce the correct curved surface behavior. We will not recap here the validations that have been performed on the method (see e.g. [MBR11] for a recent example), and will instead introduce relevant details when we later construct our 3D fractional Laplacian.

The CPM is not the first embedding method proposed for implicit surfaces, as variational versions have been available for some time [BCOS01, Gre06]. Unlike the variational versions, the CPM does not require the underlying PDEs to be rewritten to include tangent plane projections that constrain the dynamics to level sets near the interface. Greer [Gre06] described degeneracies that can occur if the narrow band boundary conditions are not carefully set in a variational method, but no such non-physical boundary conditions are needed by the CPM. Even if a variational version is preferred, all existing embedding methods require 3D generalizations for their 2D operators, so the results presented in §5.5.3 are still needed. All of the methods require the construction of extension fields, so the closest point transform we describe in §5.5.5 could also be used to accelerate the variational approaches.

5.5.2 The iWave Algorithm

The iWave algorithm [Tes04b] produces more realistic water wave behavior than alternatives such as the traditional wave equation, and is used extensively in production (see e.g. [Car07, FH09, AAB+11]). It is derived from the linearized Bernoulli's equation for irrotational flow,

$$\frac{\partial \phi}{\partial t} = -p - U, \tag{5.18}$$

where ϕ is the velocity potential, p is the pressure, and U is the potential energy. It can be stated in undamped form as the equation:

$$\frac{\partial^2 h}{\partial t^2} = -g\sqrt{-\nabla^2}h. \tag{5.19}$$

Here, h is the fluid height, t is time, g is the gravity magnitude, and $\sqrt{-\nabla^2}$ is a *fractional Laplacian* operator [Pod99, MR93]. Aside from the radical, it is very similar to the traditional wave equation, $\partial^2 h/\partial t^2 = c\nabla^2 h$. The fractional term arises because the gradient of the potential ϕ in Bernoulli's equation is constrained to be divergence-free, $\nabla^2\phi = 0$, and a squaring term in the vertical direction h must be accounted for. For this reason, it is also referred to as the *vertical derivative* operator. For further details, see §3.2 in [Tes04b].

The fractional nature of the operator significantly complicates its spatial discretization, because fractional derivatives usually have non-local support, and the resulting operator is divergent in frequency space due to the k^2 term, where k denotes the spatial frequency. Tessendorf [Tes04b] addresses the first problem by imposing a hard spatial cutoff, and the second by introducing a Gaussian "soft-cutoff" that suppresses the growth of the k^2 term. The final vertical derivative operator $G_{2D}(r)$ is then stated in polar coordinates [Tes08] as

$$G_{2D}(r) = \int_0^\infty k^2 e^{-k^2} J_0(kr)dk, \tag{5.20}$$

where r is the radial coordinate, e^{-k^2} is the soft-cutoff, and J_0 is the zeroth Bessel function of the first kind. The hard spatial cutoff is realized by only evaluating Eqn. 5.20 out to a user-specified r. Eqn. 5.20 is discretized into a convolution kernel using Algorithm 2.

Algorithm 2: iWave2DKernel(W, k_M)

Data: iWave2D is the convolution kernel, W is the spatial width of the desired kernel, and k_M is the maximum desired wave frequency to be captured.

```
begin
    iWave2D = 0
    h = ⌊W/2⌋
    for y = -h to h do
        for x = -h to h do
            r = √(x² + y²)
            for k = 0 to k_M do
                iWave2D(x,y) += k² e^(-k²) J_0(kr)
```

5.5.3 Building a 3D Vertical Derivative

In order to simulate iWave on a surface using the CPM, we need a 3D version of Eqn. 5.20 and Algorithm 2. However, unlike the Laplacian operator, the vertical derivative has no obvious 3D analog. Indeed, the definition of the operator seems to be inherently surface-based, as the radical arises from taking the square root in the normal direction. The salient

spatial function in Eqn. 5.20 is the J_0 Bessel function of the first kind, so a reasonable first attempt is to replace it with the *spherical* Bessel function of the first kind, $j_0(r) = \frac{\sin r}{r} = \text{sinc}(r)$. We found that generating a 3D kernel using a simple, naïve replacement of J_0 with j_0 results in an unstable simulation and unusable results. More care is clearly needed in the construction of the operator. The broader question is: what makes a good CPM operator? Ruuth and Merriman [RM08] reason that if u_{3D} is an extension of the scalar field u_{2D}, then u_{3D} does not vary in the normal direction, and so at the surface,

$$\nabla u_{3D} = \nabla_S u_{2D}, \tag{5.21}$$

where ∇_S denotes the 2D surface gradient. Therefore, u_{3D} will only vary along the surface, and the ∇_S operator will only induce motion in the surface tangent directions.

We examine this intuition in a slightly different form. Say we have the scalar field $u_{2D}(x,y)$, and its extension $u_{3D}(x,y,z)$. We can state Eqn. 5.21 in terms of a convolution about the origin with an arbitrary operator D:

$$\int_{x,y} D_{2D}(x,y)\, u_{2D}(x,y)\, dx\, dy = \int_{x,y,z} D_{3D}(x,y,z)\, u_{3D}(x,y,z)\, dx\, dy\, dz.$$

Since u_{3D} is an extension field, it must be constant in some normal direction. The direction is arbitrary, but for expository purposes, let us choose the z direction:

$$\int_{x,y} D_{2D}(x,y)\, u_{2D}(x,y)\, dx\, dy = \int_{x,y} u_{3D}(x,y,0)\, dx\, dy \int_z D_{3D}(x,y,z)\, dz.$$

By construction, $\int_{x,y} u_{2D}(x,y) = \int_{x,y} u_{3D}(x,y,0)$, so if we assume that $D_{3D}(x,y,z)$ is spherically symmetric, which is reasonable given that the Laplacian and gradient operators also display this form of symmetry, this further reduces to:

$$D_{2D}(x,y) = \int_z D_{3D}(x,y,z)\, dz. \tag{5.22}$$

Eqn. 5.22 provides an answer to our original question: a good 3D CPM operator *should project down to the original 2D operator*. Simple inspection shows that this condition is met by the familiar 7-point Laplacian and gradient operators. This can be viewed as ensuring that a CPM simulation on the extension field of a 2D plane produces the same results as a straight 2D simulation.

More formally, the projection of a spherically symmetric function is an *Abel transform*. So, if a natural 3D operator is not available, we can construct one by taking the *inverse* Abel transform. Fortunately, the J_0 function is both spherically symmetric and has a known Abel transform pair [Bra99],

$$\mathscr{A}^{-1}\left(J_0(r)\right) = \frac{1}{\pi}\text{sinc}(r),$$

where $\mathscr{A}^{-1}$ denotes the inverse Abel transform. Using this relation, we can now build the inverse of the 2D iWave kernel:

$$\mathscr{A}^{-1}\left(\int_0^\infty k^2 e^{-k^2} J_0(kr)dk\right) = \frac{1}{\pi}\int_0^\infty k^3 e^{-k^2}\,\mathrm{sinc}(kr)dk. \tag{5.23}$$

Note that an extra k appears due to the $J_0(kr)$ term, and we have folded it into the k^3 term. Eqn. 5.23 can now be used to generate a 3D vertical derivative kernel, provided that the removable singularity at $\mathrm{sinc}(0)$ is properly handled. Our 3D vertical derivative operator can now be stated as:

$$G_{3\mathrm{D}}(r) = \frac{1}{\pi}\int_0^\infty k^3 e^{-k^2}\,\mathrm{sinc}(kr)dk. \tag{5.24}$$

5.5.4 Reducing Projection Error

The J_0 and j_0 functions arise from the fractional operator, so they have non-local support that falls off relatively slowly in space. Some projection error is therefore inevitable, as the 3D kernel only extends a finite amount in the normal direction (z in the preceding equations), and the integral in Eqn. 5.22 will be truncated to some subinterval of $[-\infty,\infty]$.

If a normal direction is known *a priori*, e.g. if the surface is known to be a static plane, then it is possible to correct for this error. For example, the projection error $\varepsilon(x,y)$ for a single position (x,y) in the z direction is,

$$\varepsilon(x,y) = D_{2\mathrm{D}}(x,y) - \int_{-h}^{h} D_{3\mathrm{D}}(x,y,z)\,dz, \tag{5.25}$$

where h denotes the spatial cutoff of the kernel. If $\varepsilon(x,y)$ is subtracted from an appropriate kernel cell, e.g. $D_{3\mathrm{D}}(x,y,0)$, the projection error through (x,y) would be reduced to zero. Unfortunately, the normal direction generally changes according to the liquid surface, so precomputing such corrections would introduce undesirable anisotropies into the 3D kernel.

However, it is possible to eliminate all projection error from the cell with the largest weight, the *center* cell. As the kernel is spherically symmetric, $\varepsilon(0,0)$ is the same regardless of projection direction. If $\varepsilon(0,0)$ is subtracted from the center kernel cell, the projection error through the center can be eliminated entirely. With this correction, we found that the relative error of a kernel of width 15 is 0.26% under the L_∞ norm. The complete algorithm for generating the 3D iWave kernel is now shown in Algorithm 3. In all our examples, we set $k_M = 10$.

2D Validation: We have verified that in the simple 2D planar case, our 3D kernel accurately reproduces the results of the original 2D iWave algorithm. We injected wave sources along a pre-defined curve on a planar surface, and used them as inputs to both our 3D solver and the original reference implementation of the 2D iWave solver [Tes04b]. As can be seen in Figure 5.7, as well as the accompanying video, our approach is able to accurately reproduce the results obtained with the 2D iWave algorithm. To quantify the error of our three-dimensional iWave kernel, we computed the difference between the 2D and 3D simulations. We calculated the relative L_2 error of the height fields, and found a per-timestep error between 0.15% and 0.25%.

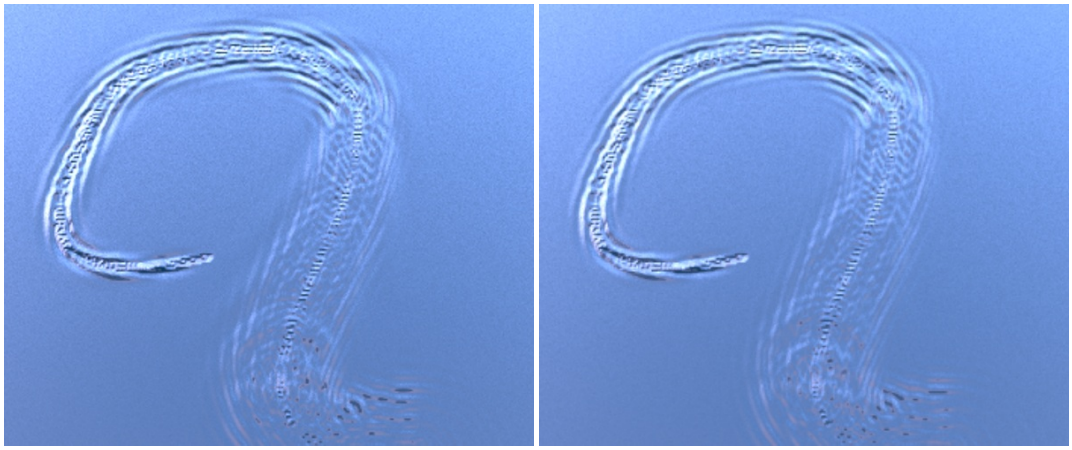


Figure 5.7: Validation of our 3D iWave kernel: On the left is the result of a standard planar 2D iWave simulation after a figure-eight shaped mouse input and 280 timesteps. On the right is the result of our 3D simulation after the same number of timesteps on the same input. Our simulation method introduces less than 1% error per timestep and produces visually indistinguishable results.

Algorithm 3: iWave3DKernel(W, k_M)

Data: W is the width of the desired kernel. k_M is the maximum desired wave frequency to be captured. iWave2D is the kernel computed by Alg. 2.

```
1  begin
2      iWave3D = 0
3      h = ⌊W/2⌋
4      for z = -h to h do
5          for y = -h to h do
6              for x = -h to h do
7                  r = sqrt(x^2 + y^2 + z^2)
8                  for k = 0 to k_M do
9                      iWave3D(x,y,z) += (k^3/π) e^{-k^2} sinc(kr)
10     sum = 0
11     for z = -h to h do
12         sum += iWave3D(0,0,z)
13     iWave3D(0,0,0) -= iWave2D(0,0) - sum
```

5.5.5 A Fast Closest Point Transform

In many previous applications of the CPM, the surface mesh is fixed, so the cost of computing the closest point transform of a surface can be amortized over many timesteps. In our application, the surface is known to be rapidly deforming, so no such amortization is possible. Therefore, it is crucial that a fast method of computing the closest point transform

be devised. One approach is to use fast marching-based methods [AS03], but this approach involves a heap search that is difficult to parallelize, and tends to smear out the scalar field. This smearing is usually considered a feature of fast marching-based methods, as it corresponds to rarefaction solutions of the Eikonal equation. In our application however, this smearing introduces spurious variations along the normal direction. The scan conversion algorithm of Mauch [Mau03] is another possibility, but it uses the surface triangle mesh, whereas we have a signed distance function that contains richer geometric information.

We have found that the signed distance function of the existing liquid simulation can be used to compute a fast, iterative, highly parallelizable closest point transform. A first order version of the algorithm is similar to the method described by Losasso et al. [LSSF]. Given a signed distance function φ, and the cell centers of the computational grid, we can compute the closest point of a cell by starting a particle, $\mathbf{c}_i$, at the cell's center, and iterating along the normal direction, $\varphi(\mathbf{c}_i) \cdot \nabla\varphi(\mathbf{c}_i)$, until $\varphi(\mathbf{c}_i) < \varepsilon$. We used the value $\varepsilon = 10^{-6}$ in our computations.

The *Nacelle* algorithm of Tessendorf [Tes11] describes a second order method of warping one level set onto another. We can use the same method to compute the closest point transform, which corresponds to a warp of all points to the zero level set. The iteration for each particle $\mathbf{c}_i$ then becomes,

$$\begin{aligned}
\Gamma &= \left(\nabla\varphi(\mathbf{c}_i)^T H(\varphi(\mathbf{c}_i))\nabla\varphi(\mathbf{c}_i)\right)/|\nabla\varphi(\mathbf{c}_i)| \\
\Delta &= -\varphi(\mathbf{c}_i)/|\nabla\varphi(\mathbf{c}_i)| \\
\mathbf{c}_i &= \mathbf{c}_i + \left(-1 + (1 + 2\cdot\Delta\cdot\Gamma)^{\frac{1}{2}}\right)\frac{\nabla\varphi(\mathbf{c}_i)}{\Gamma},
\end{aligned}$$

where $H(\cdot)$ denotes the Hessian operator. The Hessian can approach zero in flat regions of the distance field and become problematic near the medial axis, so we test if $\Gamma < \varepsilon$ at the beginning of each iteration, and fall back to first order iteration if the condition is true. We found that this variant of the *Nacelle* algorithm is highly parallelizable, and very fast. Results obtained via fast marching and our *Nacelle* variant are shown in Figure 5.8.

We found that implementing our *Nacelle* variant was quite simple, as it is essentially a particle iteration augmented by a second order correction. None of the heaps or quadratic solves involved in fast marching are needed, and the final code is drastically simpler than the canonical implementation of the Mauch algorithm [1]. Unlike fast marching methods, it also supports efficient lazy evaluation: the extension value of any random cell can be queried and computed in $O(1)$ time without computing the values at all of the intervening cells between the queried cell and the interface. We exploit this feature when performing MacCormack advection in §5.5.6.

5.5.6 Building and Advecting the Extension Field

A *Frozen Core* Extension Field: Once the closest point transform has been computed, a method must be selected to extend surface values into the narrow band. Ruuth and Merriman [RM08] originally used 4th order Newton divided differences, but subsequent work derived 4th and 6th order Weighted Essentially Non-Oscillatory (WENO) schemes [MR09], which

[1] `https://bitbucket.org/seanmauch/stlib`.

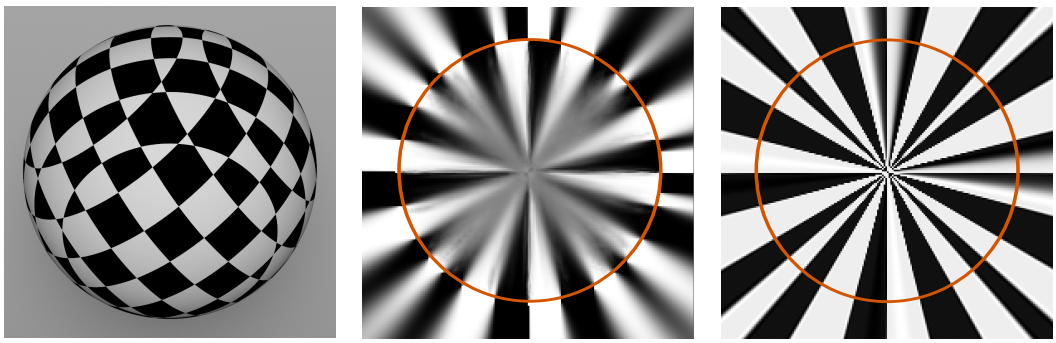
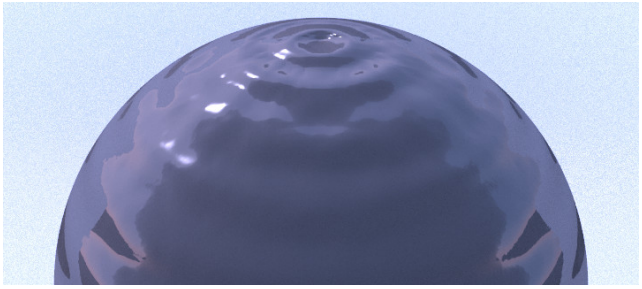


Figure 5.8: Left to right: a sphere with a checkerboard surface; the center slice of the sphere's extension field, computed using fast marching; the same slice computed using our *Nacelle* variant. Our variant is faster and does not smear out the scalars. Computing the 200^3 extension field took 3 minutes and 20 seconds with fast marching, and 21 single-threaded seconds with our algorithm.

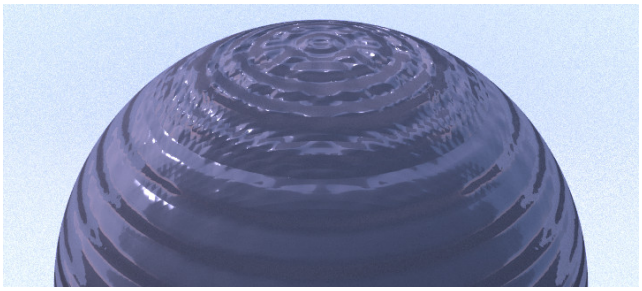
we will refer to as WENO4 and WENO6. Too frequent re-extension can be computationally expensive and smear out the scalar field unnecessarily. Greer [Gre06] observed that this is analogous to the well-known problems of periodic velocity field re-extension and signed distance field reinitialization. We also encounter severe smearing when re-extending every timestep (Figure. 5.9(a)), even when using WENO4 or WENO6. Never re-extending the surface scalars captures crisper features (Figure. 5.9(b)), but it becomes unclear if valid surface dynamics are being simulated. In both cases, undesirable anisotropies appear that reveal the underlying grid.

We found that a subtle modification fixes both of these problems. When computing the extension field, we 'freeze' the values that are less than one grid cell away from the interface. These values define the on-surface solution, and are accurately computed by the solver, so they should be smeared out as little as possible. We refer to this as a *frozen core*, as it freezes the values at the core of the narrow band. This change significantly improves the crispness of the results, and also suppresses the appearance of grid anisotropies, as shown in Fig. 5.9(c). This strategy is not entirely novel, as Adalsteinsson and Sethian [AS03] make use of a similar technique when initializing their fast marching method. However, we have not seen these substantial improvements noted anywhere else in the literature, so they are worth emphasizing here.

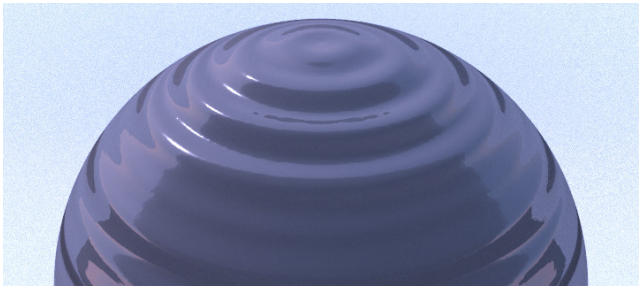
Narrow Band MacCormack Advection: We found that first order semi-Lagrangian advection [Sta99] smeared out details captured by the CPM, and opted instead to use the MacCormack advection scheme of Selle et al. [SFK+08]. Two modifications significantly improved our results. First, we replaced the linear interpolant for the backtraces with the same WENO4 scheme used for extension field construction. Second, we observed that extending and advecting the surface field unnecessarily interpolated the field *twice*: once during extension, and again during advection. In order to remove this unnecessary smearing, we construct the extension field using *nearest neighbor* interpolation. We use the *Nacelle* algorithm to find the nearest surface point, but instead of interpolating grid values to obtain a final result, we simply grab the value from the nearest grid cell. This essentially computes an anti-rarefaction solution that suppresses all variation in the normal direction. As interpo-



(a) With extension



(b) Without extension



(c) With *frozen core* extension

Figure 5.9: We inserted a small circular wave at the top of a sphere and simulated 500 timesteps using different re-extension strategies. *Top to bottom:* with WENO4 re-extension every timestep, with no re-extension, and WENO4 re-extension using the *frozen core* from §5.5.6 every step. The frozen core result does not smear out the waves and suppresses grid anisotropies that ruin the symmetry of the other cases.

lation still occurs during advection, the field is still smoothed, and we did not observe any stairstepping artifacts. In addition to producing significantly crisper surface details (see Figure 5.10), the removal of the additional WENO4 call makes this approach computationally cheaper.

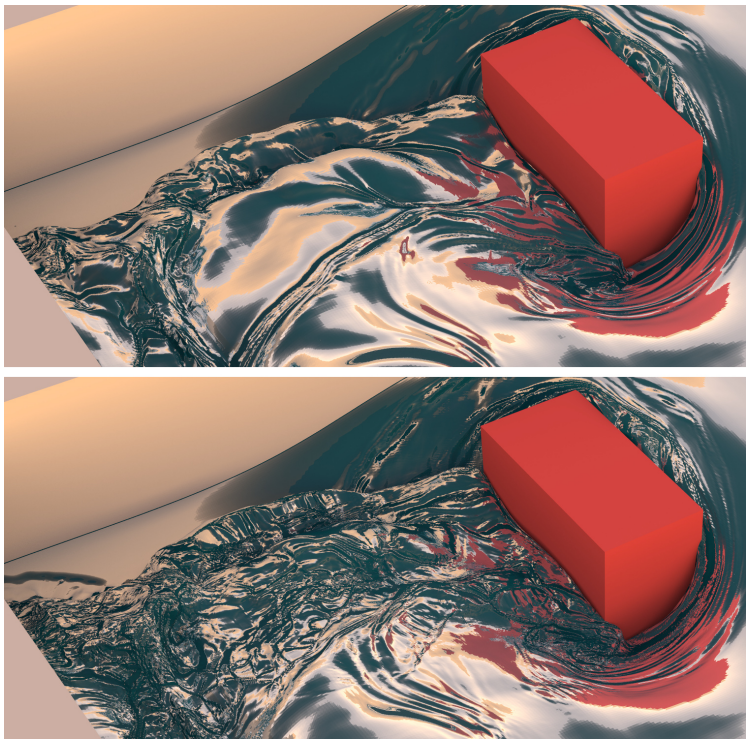


Figure 5.10: **Top:** Without *frozen core* extension and improved advection from §5.5.6. **Bottom:** Same frame, with modifications from §5.5.6. Note that the turbulent wake behind the block is lost entirely without our improvements.

The advection scheme must support narrow banding to avoid introducing a volumetric bottleneck into the algorithm. Narrow banding with first order semi-Lagrangian advection is straightforward, as backtraces can be computed for a band around the interface, and the extension values for this band can be computed on-the-fly using the *Nacelle* algorithm. However, the MacCormack method advects the field forward and backwards to compute an error term. The narrow bands for these stages differ, and computing the values for the backwards stage is significantly more complex: they correspond to an *advected* extension field, not just an extension field. Therefore, we cannot obtain valid values for the backwards band by simply applying the *Nacelle* algorithm. A obvious solution to this problem would be to fatten the narrow band for the forward stage according to the grid velocities, but in practice this results in a significant amount of unnecessary computation.

We instead performed a preliminary pass to determine the exact set of cells needed to perform narrow band MacCormack advection. We traced the velocities forward to find all the cells needed for the forward band, and traced *these* cells backwards to determine the cells needed in the backwards band. We then constructed the extension field for *all* of these

cells and performed the advection. The preliminary pass did not contain any calls to the WENO4 interpolant, and thus consisted mostly of fast integer operations that consumed 5% to 6% of the running time. By comparison, when we fattened the narrow band according to cell velocities, it doubled the extension field building time. This build time becomes the main bottleneck at high resolutions (See Table 5.1), so this approach added at least an additional 25% to the running time. Our two pass method is clearly more efficient.

5.5.7 Turbulence Seeding

Wave propagation is only perceived as realistic if waves are seeded in visually expected regions. Following the intuition of Kim et al. [KTJG08], we identify under-resolved regions of the fluid surface where details are being lost. For liquids, this corresponds to regions of high surface curvature, so we inject turbulence into locations where the absolute principal surface curvatures are close to the Nyquist limit of the current grid. This is in line with other curvature-based strategies in liquid simulations, such as those employed recently by Thüery et al. [TWGT10a] and Yu et al. [YWTY12] to seed a (classical) surface wave simulation, as well as the particle seeding strategy of Foster and Fedkiw [FF01]. Intuitively, this corresponds to regions where the 'surface skin' layer of the liquid tears and initiates surface oscillations.

We compute a source field for injecting surface waves by filtering the maximum curvature values with a Catmull-Rom spline centered at half the grid resolution, and a falloff value of one-fifth the grid resolution. Once the seeding regions have been located, we set the source term in these regions to the local Gaussian curvature in order to reflect the variations occurring along the surface. We found that the curvature computation method from Museth et al. [MBWB02] provided smooth, robust results for both the principal and Gaussian curvatures. Note that all of these quantities can be computed efficiently on the low-resolution grid.resampling to higher resolution.

We found that in some scenarios, adding this source field to the height field was sufficient (Figs. 5.12 and 5.14). However, if the appearance of a higher apparent surface resolution is desired, convolving the source field once with the vertical derivative operator creates the impression that scattering has occurred across a wider range of scales, and produces higher frequency waves. For this additional convolution, we use a vertical derivative operator (Eqn. 5.24) integrated over the $[2, k_M)$ domain, and use the extension field of the result. This seeding method was used in Figure 5.11. We exclude the $[0, 2)$ range because these frequencies are close to the Nyquist frequencies already present on the grid. We add the source field to the height fields of both the current and previous timesteps in order to convey the impression that the waves have persisted for some time, but are currently scattering into higher frequencies. Otherwise, the sources induce instantaneous velocities that produce visual spikes in the height field. We have found that these two turbulence seeding strategies, Gaussian curvature and convolved Gaussian curvature, work well in practice. These are by no means the only strategies possible, but we leave further exploration to future work.

The seeding strategy should be made aware of internal obstacles in order to avoid injecting spurious turbulence. In Figure 5.11, a large amount of surface curvature exists where the liquid wraps around the central column. Much of this curvature is along the liquid-*obstacle* interface, not the liquid-air interface, so injecting turbulence in these regions is incorrect,

and can result in overly lively waves around the column. This problem is addressed by zeroing out the source term in regions surrounding internal obstacles.

5.5.8 The Complete Algorithm

We have now described all of the components of our algorithm. The complete algorithm is shown in Algorithm 4. The extensions performed on Line 2 are the nearest neighbor extensions described in §5.5.6, while the extensions on lines 6 and 8 use the WENO4 interpolant. As the iWave algorithm uses explicit integration, the surface wave simulation is run for T user-specified substeps for every step of the coarse simulation. We found that setting $T = 5$ worked well in all of our examples. Line 11 encodes the Leapfrog scheme from Tessendorf [Tes04b].

Algorithm 4: surfaceWaves(φ^t, $\mathbf{v}^t$, CP^{t-1}, h^t, h^{t-1})

Data: φ^t and $\mathbf{v}^t$ are the current level set and velocity field of the coarse simulation; CP^t is the closest point transform of timestep t, h^t is the surface height at timestep t; α is a damping coefficient, T is the number of substeps, Δt is the surface simulation timestep size.

1 **begin**
2 $h^t = CP^{t-1}(h^t)$, $h^{t-1} = CP^{t-1}(h^{t-1})$
3 Advect h^t and h^{t-1} using $\mathbf{v}^t$.
4 Build closest point transform of φ^t, CP^t.
5 source = filtered curvature of φ^t, convolved by Eqn. 5.24.
6 source $= CP^t$(source)
7 **for** *i = 1 to T* **do**
8 $h^t = CP^t(h^t)$, $h^{t-1} = CP^t(h^{t-1})$
9 $d = h^t$ convolved by Eqn. 5.24
10 temp $= h^t$
11 $h^t = \frac{(2-\alpha\Delta t)h^t - h^{t-1}}{1+\alpha\Delta t} + \text{source} - d$
12 $h^{t-1} = \text{temp} + \text{source}$
13 Clear the source field if $i = 1$

5.6 Discussion and Results

The iWave algorithm uses the 'deep water' approximation $h \gg \lambda$, where h is the water depth and λ is the wave length. As we are simulating high frequencies, our λs are quite small, so this approximation is valid even if the liquid is globally shallow. We note that this 'relatively deep' approximation is fairly common in the fluid mechanics literature, and some practitioners [Joh97] prefer to use the terms 'short' and 'long' waves in lieu of 'deep' and 'shallow' in order to avoid any confusion. If alternate dispersion relations are desired, Eqn. 5.24 can be scaled using the exact same terms described in Tessendorf [Tes04b].

Implementation: We ran all of our simulations on a 12-core 2.66 Ghz Mac Pro with 96 GB of memory. All of our simulations fit into memory, so we did not need to use a

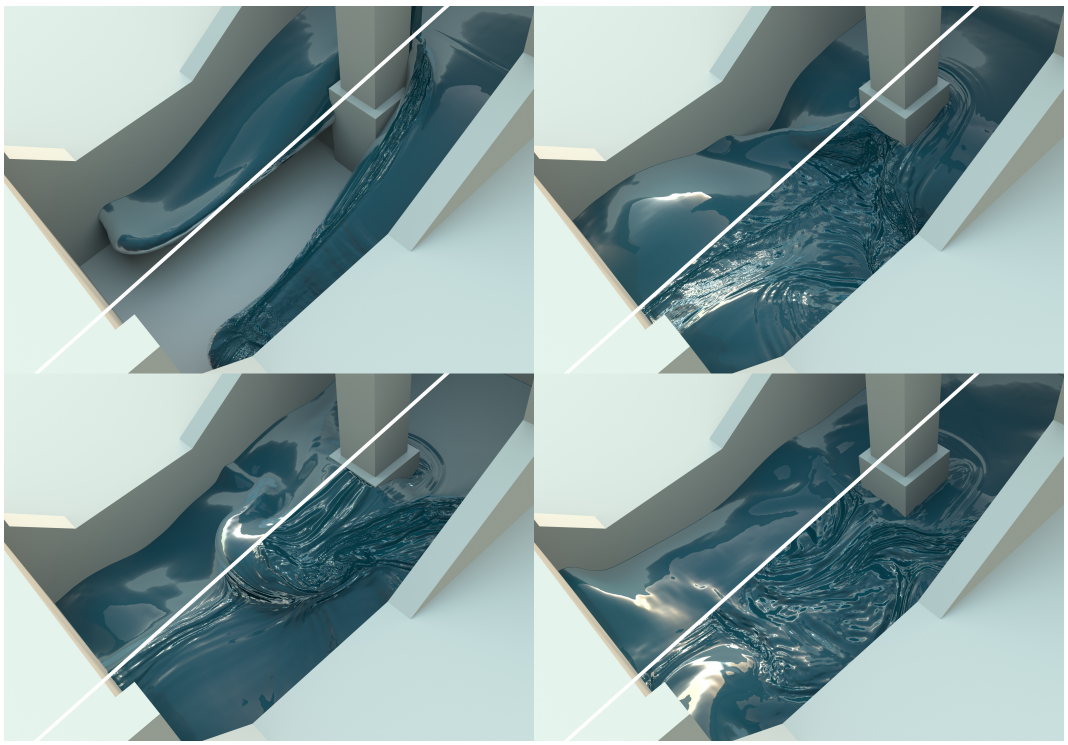


Figure 5.11: A $100^2 \times 50$ PhysBAM simulation, up-resed to $800^2 \times 400$. The left half of each image shows the original simulation, and the right shows our up-resed version. Our simulation took roughly $10\times$ the time of the original, whereas a brute force simulation would take roughly $512\times$, i.e. 8^3, the time. The source and height values around the column were clamped to zero as respectively described in §5.5.7 and §5.6.

hierarchical or blocked data structure. However, we expect that such structures would yield additional speedups due to improved memory locality. We used the WENO4 interpolant [MR08], which has a stencil width of four. The more expensive WENO6 was also tested, but it did not improve the results sufficiently to justify the additional computation. We used a 3D iWave kernel (Eqn. 5.24) with a stencil width of 15. Reducing its spatial extent would reduce the running time, but at the cost of reduced wave propagation speeds. We used OpenMP to parallelize the convolution, *Nacelle* computation, and extension field computation stages of our algorithm. While we parallelized the most computationally intensive functions, the entire algorithm can be run in parallel, so it is an excellent candidate for GPU acceleration. We found that the obstacle interaction method of the original iWave algorithm, which set the height values on the interior of obstacles to zero, worked well in our 3D generalization. No additional considerations were needed. All of our results were rendered using a modified version of PBRT [PH10] that read in the height fields from our simulations as solid textures and then used them as bump maps. The solid texture lookup function in PBRT was modified to use the WENO4 interpolant. meshes had to be subdivided impractically small to capture all the details contained in the simulation data, and that bump mapping combined with Fresnel reflections already effectively captured these details.

Houdini test: We compared both the scalability and quality of our algorithm to the Houdini simulation from Lait [Lai11]. We used the Houdini 12 solver, which utilizes a

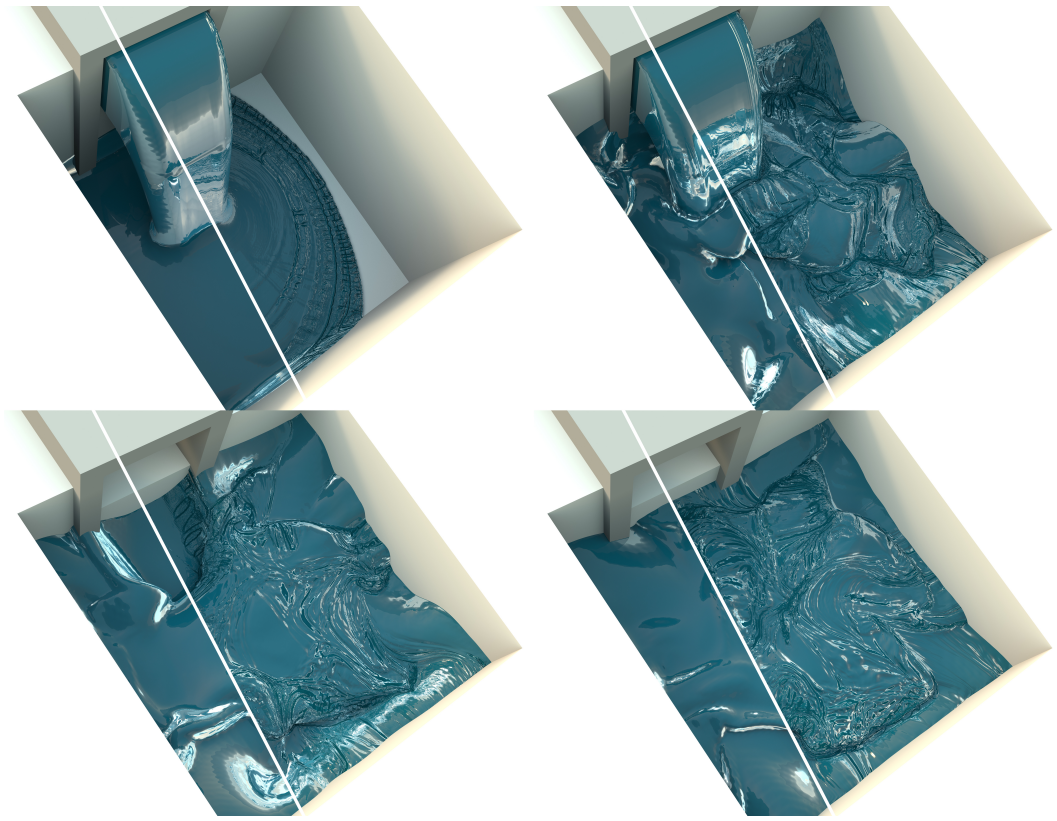


Figure 5.12: A 100^3 PhysBAM simulation, up-resed to 800^3. The left half of each image shows the original simulation, and the right shows our up-resed version. The $8\times$ up-resing only took approximately twice the time of the original simulation.

parallel Preconditioned Conjugate Gradient solver, and collected timing information for the scene at the resolutions of 100^3, 200^3 and 400^3. At the highest resolution, the simulation took nearly a week, which is totally impractical for production. The overall motion of the liquid clearly changed between resolutions.

By comparison, our solver was able to up-res the 100^3 simulation to 200^3 in *less than half of the time of the base simulation*. When comparing our results to the direct 200^3 solution, we observed that our algorithm captured higher frequency motions (Figure 5.14). The main bottleneck in the simulation at this resolution was writing the large volume files to disk, which took up 46% of the running time. Using a sparse volume data structure, or an integrated simulation-renderer solution, would yield additional speedups. We also up-resed the 100^3 simulation to 400^3, 800^3, and 1000^3. The 800^3 and 1000^3 simulations in particular captured extremely detailed surface motion, and would take months for the Houdini 12 solver to compute. Disk I/O remained one of the bottlenecks, respectively taking 25%, 22%, and 21% of the running times. Convolution and extension field construction times also become more significant, which suggests that more aggressive parallelization could yield further speedups.

The running time exhibits inferior scaling when increasing from 400^3 to 800^3, though the scaling is still significantly better than the greater than $8\times$ scalings observed for the

Example	Base Res.	Upped Res.	Frame Time	Total Time	Scaling
Houdini	100^3	N/A	00:00:24	01:39:00	-
Figure 5.14	200^3	N/A	00:03:44	14:59:00	9.08
	400^3	N/A	00:38:00	152:00:00*	10.1
	100^3	200^3	00:00:12	00:48:14	-
	100^3	400^3	00:00:58	03:55:00	4.87
	100^3	800^3	00:05:29	21:56:15	5.60
	100^3	1000^3	00:11:12	44:51:56	2.07
Pouring	100^3	N/A	00:01:52	06:13:20	-
Figure 5.12	100^3	200^3	00:00:07	00:24:38	-
	100^3	400^3	00:00:34	01:53:55	4.62
	100^3	800^3	00:03:15	13:00:52	6.85
Dam	$100^2 \times 50$	N/A	00:00:18	01:33:00	-
Break	$100^2 \times 50$	$200^2 \times 100$	00:00:06	00:30:44	-
Figure 5.11	$100^2 \times 50$	$400^2 \times 200$	00:00:21	01:45:33	3.43
	$100^2 \times 50$	$800^2 \times 400$	00:02:48	11:13:09	6.38

Table 5.1: All timings are in **hours:minutes:seconds**. The *Scaling* column denotes the scaling relative to the timing in the preceding row. Rows marked *N/A* in the *Upped Res.* column are timings for direct Houdini or PhysBAM simulations.

direct volumetric solvers. The disk I/O does not appear to be solely responsible for this, as the convolution and extension field stages also exhibit roughly this scaling. Most likely the memory traffic from the large volumes is saturating the bus, which further suggests that investigating high bandwidth architectures such as a GPU might be fruitful.

PhysBAM tests: Our algorithm is agnostic to the source of the level set data, so Figures 5.11 and 5.12 show the results of running our algorithm on two simulations produced using the PhysBAM code release [DHF+11]. In keeping with our goal of up-resing, we again ran the simulation at relatively coarse 100^3 and $100^2 \times 50$ resolutions. The simulations were run single-threaded, as the multi-threaded version of the PhysBAM release is listed as "experimental". We expect that a multi-threaded implementation would produce timings competitive with the Houdini solver.

We observed timing breakdowns and scalings that were similar to the Houdini example. Disk I/O dominates initially, but decreases to roughly a quarter of the running time at higher resolutions. The same decrease in scaling at 800^3 is also observed. In the "Dam Break" example (Figure 5.11), we were able to up-res the simulation by a factor of four along each spatial axis using approximately the same amount of time as the original simulation, and in the "Pouring" example (Fig. 5.12), in *less* time than the original simulation.

Other wave kernels: Once the components of our turbulence algorithm are in place, it becomes straightforward to experiment with other models of wave motion. We ran the "Dam Break" example using the traditional wave equation instead of the iWave kernel in Figure 5.13. The traditional wave equation still gives useful results, but the smaller kernel introduced a smaller timestep size, and the final results tended to suppress higher-frequency

waves. However, if the user prefers this 'look', we found that it could be achieved with minimal code modification. Other models, such as the Korteweg-de Vries and non-linear Schrödinger equations [Joh97] could also be used to achieve alternate looks.

The importance of damping: We found that the damping parameter, α in Algorithm 4, had a significant impact on the quality of the final results. For less damped simulations, $0 < \alpha < 0.2$, the waves persisted longer than expected and produced a distracting "memory" effect. Higher dampings, $0.2 < \alpha < 0.4$, produced behavior more in line with perceptual expectations. This is in agreement with the default setting of $\alpha = 0.3$ in the original 2D iWave implementation [Tes04b]. A comparison of various α settings can be seen in Figure 5.15 and the supplemental video.

5.7 Conclusions and Future Work

We have described an efficient, closest point method of increasing the apparent spatial resolution of an existing liquid simulation. We have addressed two main obstacles to performing this in a Eulerian setting: the construction of a 3D iWave operator, and the efficient extension of surface scalars. We have additionally described methods for maintaining simulation details, and proposed two turbulence seeding methods. The algorithm can produce surface features in running times competitive with, and sometimes superior to, the original base simulation. We have used our algorithm to simulate liquids containing detailed, high frequency motion that, to our knowledge, have not been captured by any previous method.

Since we are dealing specifically with the problem of "up-resing" a liquid, our algorithm only performs a one-way coupling. It remains to be seen if the higher frequency detail can be coupled back to the coarse simulation, as was done in Thürey et al. [TWGT10a]. Our algorithm only requires signed distance and velocity fields as inputs, so it could be applied to animated meshes, as was done in [ATBG08], by computing the signed distance field of the mesh and extrapolating velocities from the vertices. For particle-based liquid simulations, a distance function such as the one proposed by Zhu and Bridson [ZB05] could provide a basis for our approach. While an implicit version of the CPM [MR09] would allow our algorithm to take larger timesteps, modifications would be needed to the Leapfrog scheme used by the iWave algorithm. Such a scheme could significantly improve the efficiency of simulating high phase-velocity capillary waves. In §5.5.3, we imposed a physically consistent spherical symmetry constraint on the 3D kernel. Relaxing this assumption presents opportunities for motif-based stylizations such as those in Ma et al. [MWGZ09].

We have shown that CPM surface physics can be viewed in terms of an Abel transform, and that surface scalar extension and convolution become bottlenecks at high resolutions. There is a rich body of literature surrounding the Fourier-Abel-Hankel transform cycle [Bra99], so there may be a signal processing approach that can accelerate these stages. Finally, we have used the inverse Abel transform to generalize one non-trivial 2D operator, the fractional Laplacian, to 3D. We are confident that this methodology will be useful in making other surface-only operators compatible with the CPM.

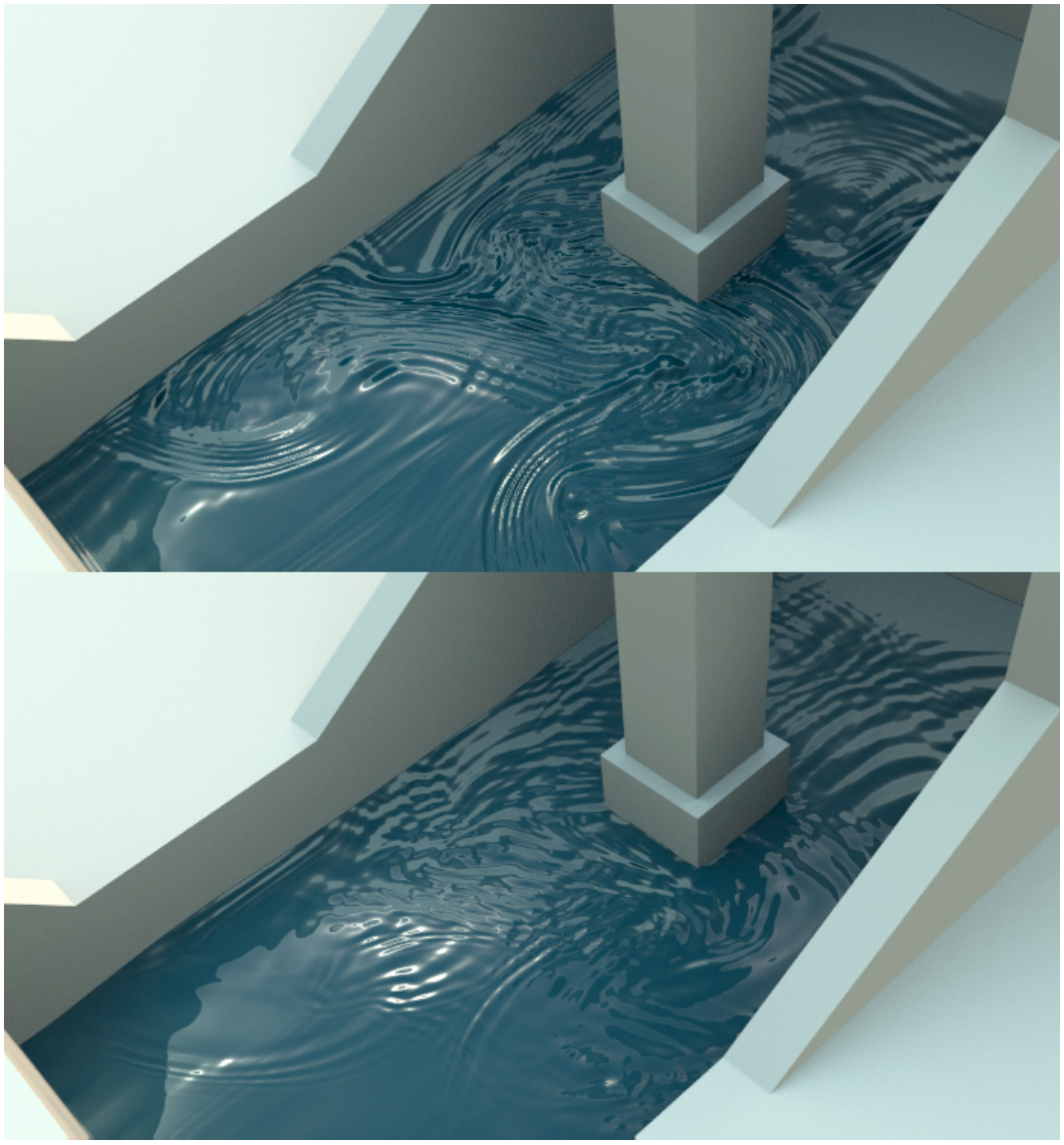


Figure 5.13: **Top:** iWave kernel **Bottom:** Traditional wave equation. When the traditional wave kernel is used, usesul results are obtained, but the waves are not as sharp, and tend towards a preferred frequency.

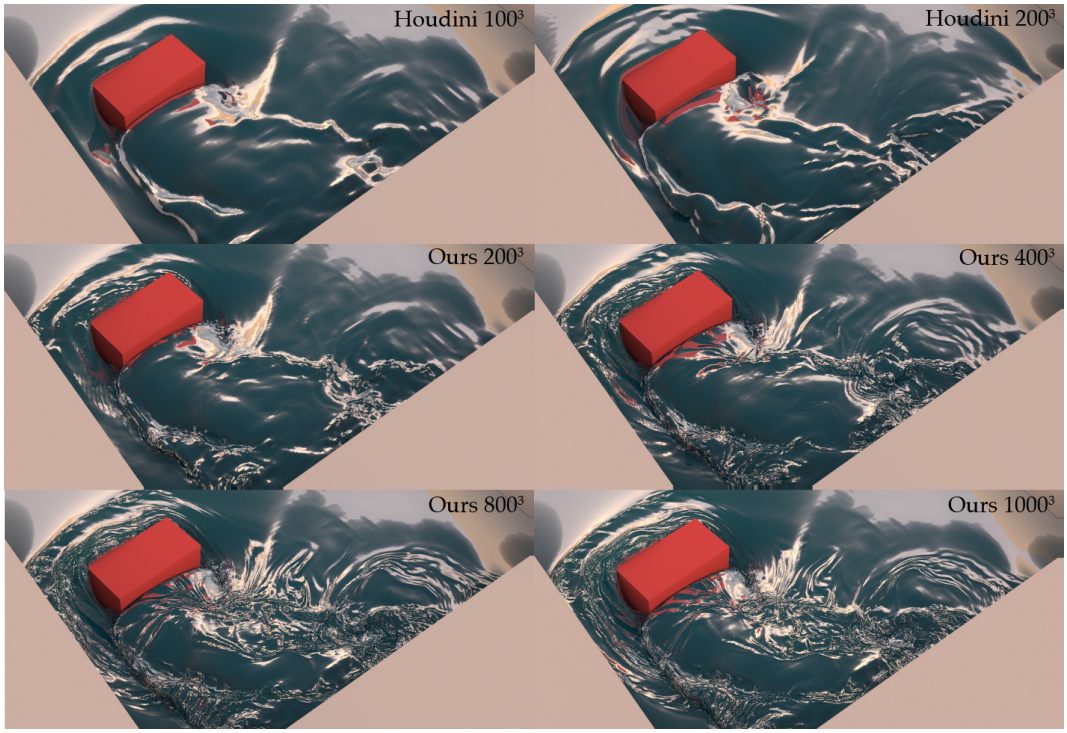


Figure 5.14: **In reading order:** Original 100^3 Houdini simulation, 2, 4, 8, and $10\times$ upres, and direct 200^3 Houdini simulation. Note how even at $2\times$ upres, higher frequency waves than those in the direct 200^3 solution are captured.

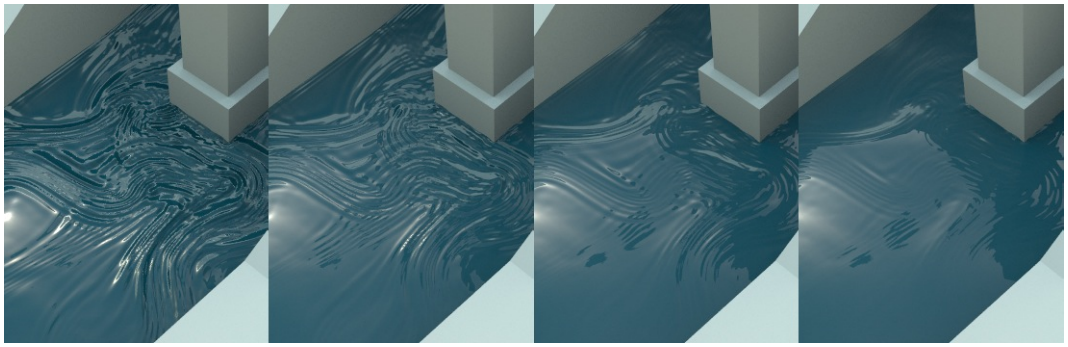


Figure 5.15: Different settings for the damping α. **Left to right:** $\alpha = 0.1, 0.2, 0.3$ and 0.4. Waves persist for too long for low dampings, but can die off too quickly before contributing any detail with high dampings.

Bibliography

[AAB+11] Alexis Angelidis, Josh Anon, Gary Bruins, Jon Reisch, and Esdras Varagnolo. Ocean mission on Cars 2. In *ACM SIGGRAPH 2011 Talks*, pages 17:1–17:1, 2011.

[AM89] M. E. Agishtein and A. A. Migdal. Dynamics of vortex surfaces in three dimensions: Theory and simulation. *Physica D*, 40:91–118, 1989.

[AMT+12] S. Auer, C. MacDonald, M. Treib, J. Schneider, and R. Westermann. Real-time fluid effects on surfaces using the Closest Point Method. *Computer Graphics Forum*, (in press), 2012.

[AN05] Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *ACM SIGGRAPH / EG Symposium on Computer Animation*, 2005.

[ANSN06] Alexis Angelidis, Fabrice Neyret, Karan Singh, and Derek Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *ACM SIGGRAPH / EG Symposium on Computer Animation*, 2006.

[AS03] D. Adalsteinsson and J. Sethian. Transport and diffusion of material quantities on propagating interfaces via level set methods. *Journal of Computational Physics*, pages 271–288, 2003.

[ATBG08] Roland Angst, Nils Thuerey, Mario Botsch, and Markus Gross. Robust and Efficient Wave Simulations on Deforming Meshes. *Computer Graphics Forum*, 27 (7):1895–1900, October 2008.

[Aup04] B. Aupoix. Modeling of compressibiliy effects in mixing layers. *Journal of Turbulence*, 5, 2004.

[BB09] T. Brochu and R. Bridson. Animating smoke as a surface. *SCA posters*, 2009.

[BBB07] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics*, 26(3):Article 100, 2007.

[BBB10] Tyson Brochu, Christopher Batty, and Robert Bridson. Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph.*, 29(4):47:1–47:9, July 2010.

[BCOS01] Marcelo Bertalmío, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys.*, 174(2):759 – 780, 2001.

[BGOS06a] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*, 25(1), 2006.

[BGOS06b] Adam W. Bargteil, Tolga G. Goktekin, James F. O'brien, and John A. Strain. A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25:19–38, January 2006.

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM SIGGRAPH papers*, 26(3):Article 46, 2007.

[BKB12] Tyson Brochu, Todd Keeler, and Robert Bridson. Linear-time smoke animation with vortex sheet meshes. In *ACM SIGGRAPH / EG Symposium on Computer Animation*, 2012.

[BL78] B. S. Baldwin and H. Lomax. Thin Layer Approximation and Algebraic Model for Seperated Turbulent Flows. *American Institute of Aeronautics and Astronautics Journal*, 1978.

[BLP98] M. Brady, A. Leonard, and D. I. Pullin. Regularized vortex sheet evolution in three dimensions. *J. Comput. Phys.*, 146:520–545, 1998.

[BM82] J. T. Beale and A. Majda. Vortex methods i: convergence in three dimensions. *Math. Comput.*, 159:1–27, 1982.

[BMWG07] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. Tracks: toward directable thin shells. *ACM Trans. Graph.*, 26(3), July 2007.

[BP01] M. Brocchini and D. H. Peregrine. The dynamics of strong turbulence at free surfaces. Part 1. description. *Journal of Fluid Mechanics*, 449:225–254, 2001.

[BP12] Alfred Barnat and Nancy S. Pollard. Smoke sheets for graph-structured vortex filaments. In *Proceedings of the 2012 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12. ACM, 2012.

[Bra99] Ronald Bracewell. *The Fourier Transform and Its Applications*. McGraw–Hill, 1999.

[Bri08] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

[BSM+06] Adam W. Bargteil, Funshing Sin, Jonathan E. Michaels, Tolga G. Goktekin, and James F. O'Brien. A texture synthesis method for liquid animations. In *ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 345–351, 2006.

[Car07] Deborah Carlson. Wave displacement effects for Surf's Up. In *ACM SIGGRAPH 2007 Sketches*, New York, NY, USA, 2007. ACM.

[CB73] A. J. Chorin and P. S. Bernard. Discretization of a vortex sheet, with an example of roll-up. *J. Comp. Phys.*, 13:423–429, 1973.

[CD05] Robert Cook and Tony DeRose. Wavelet noise. In *Proceedings of ACM SIGGRAPH 2005*, volume 25, 2005.

[Cho81] A. J. Chorin. Estimates of intermittency, spectra and blow-up in developed turbulence. *Comm. on Pure and Applied Math.*, 34:853–866, 1981.

[Cho96] A. J. Chorin. Microstructure, renormalization and more efficient vortex methods. *ESAIM Proc.*, 1:1–14, 1996.

[CK95] M. K. Chung and S. K. Kim. A nonlinear return-to-isotropy model with turbulent fluctuations. *Phys. Fluids*, 7:1425–1436, 1995.

[CK99] Georges-Henri Cottet and Petros Koumoutsakos. *Vortex Methods: Theory and Practice*. Cambridge Univ. Press, 1999.

[CLB+09] Ming Chuang, Linjie Luo, Benedict J. Brown, Szymon Rusinkiewicz, and Michael Kazhdan. Estimating the Laplace-Beltrami operator by restricting 3d functions. In *Eurographics Symposium on Geometry Processing*, pages 1475–1484, 2009.

[CM10] Nuttapong Chentanez and Matthias Müller. Real-time simulation of large bodies of water with small scale details. *Proceedings of the 2010 ACM SIGGRAPH Symposium on Computer Animation*, pages 197–206, 2010.

[CM11] Nuttapong Chentanez and Matthias Mueller. Real-time eulerian water simulation using a restricted tall cell grid. *ACM Trans. Graph.*, 30:82:1–82:10, 2011.

[CMT04] M. Carlson, P. J. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23:377–384, 2004.

[Cow73] G.R. Cowper. Gaussian quadrature formulas for triangles. *Int. J. Num. Methods*, 7(3):405–408, 1973.

[CP03] G. H. Cottet and P. Poncet. Advances in direct numerical simulations of 3d wall-bounded flows by vortex-in-cell methods. *J. Comput. Phys.*, 193:136–158, 2003.

[CS07] B. Chaoat and R. Schiestel. From single-scale turbulence models to multiple-scale and subgrid-scale models by fourier transform. *Theor. and Comp. Fluid Dyn.*, 21(3):201–229, 2007.

[CTG10] Jonathan Cohen, Sarah Tariq, and Simon Green. Interactive fluid-particle simulation using translating eulerian grids. In *Proceedings of the 2010 SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2010.

[CZY11] Fan Chen, Ye Zhao, and Zhi Yuan. Langevin particle: A self-adaptive lagrangian primitive for flow simulation enhancement. *Computer Graphics Forum*, 30(2):435–444, 2011.

[DCGG11] E. Darles, B. Crespin, D. Ghazanfarpour, and J. Gonzato. A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics. *Comput. Graph. Forum*, 30:43–60, 2011.

[Deh02] Walter Dehnen. A hierarchial o(n) force calculation algorithm. *J. Comput. Phys.*, 179:27–42, 2002.

[dFN01] Javier de Frutus and Julia Novo. A spectral element method for the navier-stokes equations with improved accuracy. *SIAM journal on Numerical Analysis*, 38(3):799–819, 2001.

[DGS07] J. Dandois, E. Garnier, and P. Sagaut. Numerical simulation of active separation control by a synthetic jet. *J. Fluid Mech.*, 574:25–58, 2007.

[DHF+11] Pradeep Dubey, Pat Hanrahan, Ronald Fedkiw, Michael Lentine, and Craig Schroeder. PhysBAM: physically based simulation. In *ACM SIGGRAPH 2011 Courses*, pages 10:1–10:22, 2011.

[DJ89] P. G. Drazin and R. S. Johnson. *Solitons: An Introduction*. Cambridge University Press, 1989.

[DK99] Frédéric Dias and Christian Kharif. Nonlinear gravity and capillary-gravity waves. *Annual Review of Fluid Mechanics*, 31:301–346, 1999.

[DMG89] P. Degond and S. Mas-Gallic. The weighted particle method for convetion-diffusion equations. *Part I: The case of an isotropic viscosity*, 53:485–507, 1989.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *Proc. SIGGRAPH*, pages 317–324, 1999.

[DR77] VD Djordjevic and LG Redekopp. On two-dimensional packets of capillary-gravity waves. *Journal of Fluid Mechanics*, 79(4):703–714, 1977.

[Dri56] E. R. Van Driest. On turbulent flow near a wall. *J. Aeronaut. Sci.*, 23(11):1007–1011, 1956.

[DS74] A. Davey and K. Stewartson. On three-dimensional packets of surface waves. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 338(1613):101–110, 1974.

[Dur93] P. A. Durbin. A reynolds stress model for near-wall turbulence. *J. Fluid Mech.*, 249:465–498, 1993.

[EFFM02] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*, 183:83–116, 2002.

[EMF02] D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of ACM SIGGRAPH*, pages pp. 736–744, 2002.

[Fal10] Eric Falcon. Laboratory experiments on wave turbulence. *Discrete. Cont. Dyn.-B*, 13:819–840, 2010.

[FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proc. of SIGGRAPH*, pages 23–30, 2001.

[FH09] Lucio Flores and David Horsley. Underground cave sequence for Land of the Lost. In *ACM SIGGRAPH 2009 Talks*, pages 6:1–6:1, New York, NY, USA, 2009. ACM.

[FKPG96] M. Farge, N. Kevlahan, V. Perrier, and E. Goirand. Wavelets and turbulence. *Proceedings of the IEEE*, 84(4):639–669, 1996.

[FOK05] Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. In *Proceedings of ACM SIGGRAPH*, 2005.

[Fri95] Uriel Frisch. *Turbulence: The Legacy of A. N. Kolmogorov*. Cambridge University Press, 1995.

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH*, pages 15–22, 2001.

[GBF03] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 22(3):871–878, 2003.

[GBO04] Tolga G. Goktekin, Adam W. Bargteil, and James F. O'Brien. A method for animating viscoelastic fluids. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2004)*, 23(3):463–468, 2004.

[Gha03] A. Gharakhani. Application of vrm to les of incompressible flow. *J. Turb.*, 4:4, 2003.

[GM77] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Not. R. Astron. Soc.*, pages 375–389, 1977.

[GO93] B. Galperin and S. A. Orszag. *Large Eddy Simulations of Complex Engineering and Geophysical Flows*. Cambridge University Press, 1993.

[Gre06] John B. Greer. An improvement of a recent Eulerian method for solving pdss on general geometries. *J. Sci. Comput.*, 29(3):321–352, June 2006.

[Hal79] O. H. Hald. The convergence of vortex methods. *SIAM J. Numer. Anal.*, 32:791–809, 1979.

[HJ00] D. C. Haworth and K. Jansen. Large-eddy simulation on unstructured deforming meshes: towards reciprocating ic engines. *Computers and Fluids*, 29:493–524, 2000.

[HK03] J. Hong and C. Kim. Animation of bubbles in liquid. *Proceedings of Eurographics 2003*, 22(3), 2003.

[HK10] Nambin Heo and Hyeong-Seok Ko. Detail-preserving fully-Eulerian interface tracking framework. *ACM Trans. Graph.*, 29:176:1–176:8, December 2010.

[HMK11] Ruoguan Huang, Zeki Melek, and John Keyser. Preview-based sampling for controlling gaseous simulations. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–186, 2011.

[HN81] C. W. Hirt and B. D. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *J. Comp. Phys*, 39:201–225, 1981.

[Hog85] SJ Hogan. The fourth-order evolution equation for deep-water gravity-capillary waves. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 402(1823):359–372, 1985.

[HPP76] J. Hardy, O. De Pazzis, and J. Pomeau. Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *Physical Review A*, 13:1949–1960, 1976.

[Hsu81] C. Hsu. *A curviliniear-coordinate method for momentum, heat and mass transfer in domains of irregular geometry*. PhD thesis, University of Minnesota, 1981.

[HW66] F. Harlow and E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluids with free surface. *Physics of Fluids*, 8, 1966.

[HZQW10] Yi Hong, Dengming Zhu, Xianjie Qiu, and Zhaoqi Wang. Geometry-based control of fire simulation. *The Visual Computer*, 26, September 2010.

[IGLF06] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transactions on Graphics*, 25(3):805–811, 2006.

[IKLH04] M. Ikits, J. Kniss, A. Lefohn, and C. Hanson. *GPU Gems: Programming techniques for real-time Graphics*. Addison Wesley, 2004.

[JC94] M. Jones and M. Chen. A new approach to the construction of surfaces from contour data. *Computer Graphics Forum*, 13(3):pp. 75–84, 1994.

[JKB+10] Taekwon Jang, Heeyoung Kim, Jinhyuk Bae, Jaewoo Seo, and Junyong Noh. Multilevel vorticity confinement for water turbulence simulation. *Vis. Comput.*, 26(6-8):873–881, June 2010.

[JO93] Javier Jiménez and Paolo Orland. The rollup of a vortex layer near a wall. *Journal of Fluid Mechanics*, 1993.

[Joh97] R. S. Johnson. *A Modern Introduction to the Mathematical Theory of Water Waves*. Cambridge University Press, 1997.

[Joh06] Voker John. On large eddy simulation and variational multiscale methods in the numerical simulation of turbulent incompressible flows. *Applications of Mathematics*, 51:321–353, 2006.

[KAK+07] V. Kwatra, D. Adalsteinsson, T. Kim, N. Kwatra, M. Carlson, and M.C. Lin. Texturing fluids. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):939 –952, sept.-oct. 2007.

[KFCO06] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. In *Proceedings of ACM SIGGRAPH*, 2006.

[KLLR05] ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. Flowfixer: Using BFECC for fluid simulation. In *Proceedings of Eurographics Workshop on Natural Phenomena*, 2005.

[KLySK12] Doyub Kim, Seung Woo Lee, Oh young Song, and Hyeong-Seok Ko. Baroclinic turbulence with varying density and temperature. *IEEE Transactions on Visualization and Computer Graphics*, 18:1488–1495, 2012.

[KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of SIGGRAPH*, pages 49–57, 1990.

[Kol41] A.N. Kolmogorov. The local structure of turbulence in incompressible viscous fluid for very large reynolds number. *Dokl. Akad. Nauk SSSR*, 30, 1941.

[Kol05] Ravikrishna Kolluri. Provably good moving least squares. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 1008–1018, August 2005.

[KSK09a] Doyub Kim, Oh-Young Song, and Hyeong-Seok Ko. Stretching and wiggling liquids. *ACM Transactions on Graphics*, 28(5):120, 2009.

[KSK09b] Doyub Kim, Oh-young Song, and Hyeong-Seok Ko. Stretching and wiggling liquids. *ACM Trans. Graph.*, 28(5):120:1–120:7, December 2009.

[KTJG08] Theodore Kim, Nils Thuerey, Doug James, and Markus Gross. Wavelet Turbulence for Fluid Simulation. *ACM Transactions on Graphics (SIGGRAPH)*, 27 (3):6, August 2008.

[KySK08] Doyub Kim, Oh young Song, and Hyeong-Seok Ko. A semi-lagrangian cip fluid solver without dimensional splitting. *Comput. Graph. Forum (Proc. Eurographics)*, 27(2):467–475, 2008.

[Lai11] Jeff Lait. Correcting low frequency impulses in distributed simulations. In *ACM SIGGRAPH Talks*, pages 53:1–53:2, 2011.

[Leo75] A. Leonard. Numerical simulation of interacting, three-dimensional vortex filaments. In *Proceedings of the IV Intl. Conf. on Numerical Meth.*, 1975.

[Leo80] A. Leonard. Vortex methods for flow simulation. *J. Comput. Phys.*, 37:289–335, 1980.

[LF02] Arnauld Lamorlette and Nick Foster. Structural modeling of flames for a production environment. In *Proceedings of ACM SIGGRAPH*, 2002.

[LGF04] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *Proceedings of ACM SIGGRAPH*, pages 457–462, 2004.

[LGOD98] A. Lozano, A. Garc?a-Olivares, and C. Dopazo. The instability growth leading to a liquid sheet breakup. *Phys. Fluids*, 10(9):2188–2197, 1998.

[LK01] K. Lindsay and R. Krasny. A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow. *J. Comput. Phys.*, 172:879–907, 2001.

[LMK97] Hung Le, Parviz Moin, and John Kim. Direct numerical simulation of turbulent flow over a backward-facing step. *J. Fluid Mech.*, 330(01):349–374, 1997.

[LMLS06] R. B. Langtry, F. R. Menter, S. R. Likki, and Y. B. Suzen. A correlation-based transition model using local variables. *J. Turbomach.*, 128:123–143, 2006.

[LRR75] B. E. Launder, G. J. Reece, and W. Rodi. Progress in the development of a reynolds-stress turbulence closure. *J. Fluid Mech.*, 68:537–566, 1975.

[LS74] B. E. Launder and D. B. Sharma. Applications of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc. *Lett. Heat Mass Transf.*, 1:1031–138, 1974.

[LSSF] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw. Multiple interacting liquids. *Proceedings of ACM SIGGRAPH*.

[LZF10] Michael Lentine, Wen Zheng, and Ronald Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph.*, 29:114:1–114:9, July 2010.

[Mau03] Sean Mauch. *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations*. PhD thesis, California Institute of Technology, Pasadena, CA, USA, April 2003.

[MBR11] Colin B. Macdonald, Jeremy Brandman, and Steven J. Ruuth. Solving eigenvalue problems on curved surfaces using the Closest Point Method. *J. Comput. Phys.*, 230(22), 2011.

[MBWB02] Ken Museth, David E. Breen, Ross T. Whitaker, and Alan H. Barr. Level set surface editing operators. *ACM Trans. Graph.*, 21:330–338, July 2002.

[MCP+09] Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-Preserving Integrators for Fluid Animation. *ACM SIGGRAPH Papers*, 28(3):Article 38, Aug 2009.

[MCPN08] Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. Low viscosity flow simulations for animation. In *ACM SIGGRAPH / EG Symposium on Computer Animation*, pages 9–18, July 2008.

[Men78] J. C. S. Meng. The physics of vortex-ring evolution in a stratified and shearing environment. *J. Fluid Mech.*, 84(3):455–469, 1978.

[MG96] J. S. Marshall and J. R. Grant. Penetration of a blade into a vortex core: vorticity response and unsteady blade forces. *J. Fluid Mech.*, 306:83–109, 1996.

[MK05] F. Menter and M. Kuntz. A scale-adaptive simulation model using two-equation models. *AIAA paper 05-1095*, 2005.

[MR93] Kenneth Miller and Bertram Ross. *An introduction to the fractional calculus and fractional differential equations*. Wiley & Sons, 1993.

[MR08] Colin B. Macdonald and Steven J. Ruuth. Level set equations on surfaces via the Closest Point Method. *J. Sci. Comput.*, 35(2–3):219–240, June 2008.

[MR09] Colin B. Macdonald and Steven J. Ruuth. The implicit Closest Point Method for the numerical solution of partial differential equations on surfaces. *J. Sci. Comput.*, 31(6):4330–4350, 2009.

[MSKG05] M. Müller, B. Solenthaler, R. Keiser, and M. Gross. Particle-based fluid-fluid interaction. *ACM SIGGRAPH / EG Symposium on Computer Animation*, 2005.

[MTPS04] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23:449–456, August 2004.

[MWGZ09] Chongyang Ma, Li-Yi Wei, Baining Guo, and Kun Zhou. Motion field texture synthesis. *ACM Trans. Graph.*, 28:110:1–110:8, December 2009.

[NB11] Michael B. Nielsen and Robert Bridson. Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph.*, 30:83:1–83:8, August 2011.

[NCZ+09] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 217–226, New York, NY, USA, 2009. ACM.

[Ney03] Fabrice Neyret. Advected textures. In *ACM SIGGRAPH/EG Symposium on Computer Animation (SCA)*, 2003.

[NKL+07] Rahul Narain, Vivek Kwatra, Huai-Ping Lee, Theodore Kim, Mark Carlson, and Ming C. Lin. Feature-Guided Dynamic Texture Synthesis on Continuous Flows. In *Eurographics Symposium on Rendering*, pages 361–370, 2007.

[NSCL08] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. *ACM SIGGRAPH Asia papers*, page Article 166, 2008.

[Obu41] A.M. Obukhov. The spectral energy distribution in a turbulent flow. *Dokl. Akad. Nauk*, 32:22–24, 1941.

[OW72] J. T. Oden and L. C. Wellford. Analysis of viscous flow by the finite element method. *AIAA J.*, 10:1590, 1972.

[Pan71] S. Panchev. *Random Functions and Turbulence*. Oxford: Pergamon Press, 1971.

[Pao65] Y. H. Pao. Structure of turbulent velocity and scalar fields at large wavenumbers. *Phys. Fluids*, 8:1063–1075, 1965.

[Per85] Ken Perlin. An image synthesizer. In *Proceedings of ACM SIGGRAPH*, pages 287–296, 1985.

[PH10] Matt Pharr and Greg Humphreys. *Physically-Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2010.

[Pod99] Igor Podlubny. *Fractional Differential Equations*. Academic Press, 1999.

[Pop83] S. B. Pope. A lagrangian two-time probability density function equation for inhomogeneous turbulent flows. *Phys. Fluids*, 26:3448–3450, 1983.

[Pop00] Stephen B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[PPB95] Valerie Perrier, Thierry Philipovitch, and Claude Basdevant. Wavelet spectra compared to fourier spectra. *Journal of Mathematical Physics*, 36, 1995.

[Pra45] L. Prandtl. über ein neues formelsystem für die ausgebildete turbulenz. *Nachr. Akad. Wiss. Göttingen K1*, pages 6–10, 1945.

[PTC+10] Tobias Pfaff, Nils Thuerey, Jonathan Cohen, Sarah Tariq, and Markus Gross. Scalable fluid simulation using anisotropic turbulence particles. *SIGGRAPH Asia papers*, pages 174:1–174:8, 2010.

[PTG12] Tobias Pfaff, Nils Thuerey, and Markus Gross. Lagrangian Vortex Sheets for Animating Fluids. *ACM Transactions on Graphics (SIGGRAPH)*, 31 (4):8, August 2012.

[PTM09] S. Patel, J. Tessendorf, and J. Molemaker. Monocoupled 3D and 2D river simulations. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Posters Session*, 2009.

[PTSG09] Tobias Pfaff, Nils Thuerey, Andrew Selle, and Markus Gross. Synthetic turbulence using artificial boundary layers. *ACM Transactions on Graphics*, 28(5):121:1–121:10, 2009.

[PWS+02] P. Ploumhans, G. S. Winckelmans, J. K. Salmon, A. Leonard, and M. S. Warren. Vortex methods for direct numerical simulation of three-dimensional bluff body flows. *J. Comput. Phys.*, 178:427–463, 2002.

[QV01] L. Qian and M. Vezza. A vorticity-based method for incompressible unsteady viscous flows. *J. Comput. Phys.*, pages 172:515–542, 2001.

[RM08] Steven J. Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.*, 227(3):1943 – 1961, 2008.

[RMSG+08] Avi Robinson-Mosher, Tamar Shinar, Jon Gretarsson, Jonathan Su, and Ron Fedkiw. Two-way coupling of fluids to rigid and deformable solids and shells. *ACM SIGGRAPH papers*, 27(3):Article 46, August 2008.

[RNGF03] Nick Rasmussen, Duc Quang Nguyen, Willi Geiger, and Ronald Fedkiw. Smoke simulation for large scale phenomena. In *Proceedings of ACM SIGGRAPH*, 2003.

[Ros31] L. Rosenhead. The formation of vorticies from a surface of discontinuity. *Proc. Roy. Soc. London*, 134:170–192, 1931.

[SA94] P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA Paper*, 92:0439, 1994.

[Sav06] Ralph Savelsberg. Experiments on Free-Surface Turbulence. *Ph.D. thesis*, 2006.

[SB08a] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–7, 2008.

[SB08b] Hagit Schechter and Robert Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, 2008.

[SDT08] M. Stock, W.J.A. Dahm, and G. Tryggvason. Impact of a vortex ring on a density interface using a regularized inviscid vortex sheet method. *J. Comp. Phys.*, 227:9021–9043, 2008.

[SF93] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In *Proceedings of ACM SIGGRAPH*, 1993.

[SFK+08] Andrew Selle, Ronald Fedkiw, ByungMoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable MacCormack method. *Journal of Scientific Computing*, 2008.

[Sma63] J. Smagorinsky. General circulation experiments with the primitive equations. i. the basic experiment. *Monthly Weather Review*, 1963.

[Smi61] Oliver K. Smith. Eigenvalues of a symmetric 3×3 matrix. *Comm. of the ACM*, 4, 1961.

[Spa09] P. R. Spalart. Detached eddy simulation. *Annual Review of Fluid Mechanics*, 41:181–202, 2009.

[SR07] Philippe R. Spalart and Christopher L. Rumsey. Effective inflow conditions for turbulence models in aerodynamic calculations. *AIAA Journal*, 45(10), 2007.

[SRF05] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *Proceedings of ACM SIGGRAPH*, 24(3):910–914, 2005.

[SS00] Boris Shraiman and Eric Siggia. Scalar turbulence. *Nature*, (405):639–646, 2000.

[Sta99] Jos Stam. Stable fluids. In *Proceedings of ACM SIGGRAPH*, 1999.

[Sto06] Mark Stock. *A Regularized Inviscid Vortex Sheet Method for Three Dimensional Flows With Density Interfaces*. PhD thesis, University of Michigan, 2006.

[SvD96] S. Shankar and L. van Dommelen. A new diffusion procedure for vortex methods. *J. Comput. Phys.*, 127:88–109, 1996.

[SvdW08] Ralph Savelsberg and Willem van de Water. Turbulence of a free surface. *Physical Review Letters*, 100:034501, Jan 2008.

[SY05] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 229–236, 2005.

[TA83] G. Tryggvason and H. Aref. Numerical experiments on hele-shaw flow with a sharp interface. *J. Fluid Mech.*, 136:1–30, 1983.

[Tes04a] Jerry Tessendorf. Interactive water surfaces. In *Game Programming Gems 4*. Charles River Media, 2004.

[Tes04b] Jerry Tessendorf. Simulating ocean water. In *ACM SIGGRAPH Courses*, 2004.

[Tes08] Jerry Tessendorf. Vertical derivative math for iwave, October 2008.

[Tes11] Jerry Tessendorf. Resolution independent volumes. In *ACM SIGGRAPH Courses*, 2011.

[TIR06] N. Thuerey, K. Iglberger, and U. Rüde. Free Surface Flows with Moving and Deforming Objects for LBM. *Proceedings of Vision, Modeling and Visualization 2006*, pages 193–200, Nov 2006.

[Tre00] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. SIAM, 2000.

[TWGT10a] Nils Thuerey, Chris Wojtan, Markus Gross, and Greg Turk. A Multiscale Approach to Mesh-based Surface Tension Flows. *ACM Transactions on Graphics (SIGGRAPH)*, 29 (4):10, July 2010.

[TWGT10b] Nils Thuerey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. *ACM Trans. Graph.*, 29(4):48:1–48:10, July 2010.

[vFWTS08] Wolfram von Funck, Tino Weinkauf, Holger Theisel, and Hans-Peter Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions in Visualization and CG*, 14(6):1396–1403, 2008.

[Wan04] Qian Xi Wang. Variable order revised binary treecode. *J. Comput. Phys.*, 200:192–210, 2004.

[Wil93] D. C. Wilcox. *Turbulence modelling for CFD*. DCW Industries, 1993.

[WMKG07] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete Laplace operators: no free lunch. In *Eurographics Symposium on Geometry Processing*, pages 33–37, 2007.

[WP10] S. Weissmann and U. Pinkall. Filament-based smoke with vortex shedding and variational reconnection. *ACM Transactions on Graphics*, 29(4), 2010.

[WSW+96] G. S. Winckelmans, J. K. Salmon, M. S. Warren, A. Leonard, and B. Jodoin. Application of fast parallel and sequential tree coeds to computing three-dimensional flows with the vortex element and boundary element methods. *ESAIM Proc.*, 1:225–240, 1996.

[WTGT09] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28 (3):9, August 2009.

[WTGT10] Chris Wojtan, Nils Thuerey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. *ACM Transactions on Graphics*, 29,3:8, July 2010.

[YCZ11] Zhi Yuan, Fan Chen, and Ye Zhao. Pattern-guided smoke animation with Lagrangian coherent structure. *ACM Trans. Graph.*, 30:136:1–136:8, December 2011.

[YHK07] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Trans. Graph.*, 26, July 2007.

[YKH+09] J.-C. Yoon, H. R. Kam, J.-M. Hong, S.-J. Kang, and C.-H. Kim. Procedural synthesis using vortex particle method for fluid simulation. *Compu. Graph. Forum*, 28(7):1853–1859, 2009.

[YNBH09] Qizhi Yu, Fabrice Neyret, Éric Bruneton, and Nicolas Holzschuch. Scalable real-time animation of rivers. *Comput. Graph. Forum*, 28(2):239–248, 2009.

[YWTY12] Jihun Yu, Chris Wojtan, Greg Turk, and Chee Yap. Explicit mesh surfaces for particle based fluids. *ACM Eurographics*, 2012.

[YZC12] Zhi Yuan, Ye Zhao, and Fan Chen. Incorporating stochastic turbulence in particle-based fluid simulation. *The Visual Computer*, (in press), 2012.

[ZB05] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *Proceedings of ACM SIGGRAPH*, 24(3):965–972, 2005.

[ZLF92] V. E. Zakharov, V. S. L'Vov, and G. Falkovich. *Kolmogorov Spectra of Turbulence I: Wave Turbulence*. Springer–Verlag, 1992.

[ZYC10] Ye Zhao, Zhi Yuan, and Fan Chen. Enhancing fluid animation with adaptive, controllable and intermittent turbulence. *ACM Eurographics*, 2010.