Upsampling Fluid Simulations

Felix König*
TU Vienna
Institute of Computer Graphics and Algorithms

Abstract

This paper aims at presenting fluid simulations on an Eulerian grid and explores different strategies which improve those simulations. The first section elaborates on the building blocks for advanced algorithms. The fundamental concepts of a standard fluid solver are covered in great detail. The paper therefore also serves as an introductory reading to people that are not familiar with the basic concepts of Navier-Stokes equations and grid like numerical solvers. All advanced methods are described in the following chapters. Furthermore, the paper also discusses hybrid approaches between Lagrangian and purely Eulerian methods. The conclusion gives an outline and comparison of the presented techniques.

CR Categories: 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;1.3.5 [Computer Graphics]: Computational Geometry and Object Modeling— Physically based modeling

Keywords: fluid simulation, Navier-Stokes equations, tetrahedral discretization, adaptivity, wavelet turbulence, vorticity confinement, Lagrangian fluid simulation, procedural noise, extrapolation, PDE solvers, stable solvers

1 Introduction

The topic of simulating both visually convincing and physically realistic dynamics of fluids and smoke is important in the field of animation, cinematography and within the development of computer games. The creation of new algorithms that model complex motion of fluids depends on different factors. Computations should of course be as fast as possible and their results are expected to deliver a level of detail and complexity that is comparable to real nature phenomena. At the same time, algorithms ought to provide enough flexibility for the artist to deviate from pure physical realism but still achieve convincing animations. Recent algorithms can be divided into two classes while both classes try to add more detail into a simulation:

- Algorithms that try to improve the numerical approximations that a fluid solver makes and
- algorithms that try to provide a convincing feel of realism.

All of those classes try to achieve their results with as minimal computational overhead as possible. Especially if those algorithms are not entirely physically accurate, details are synthesized in a fashion that allows fast computation of results.

*e-mail: e0917104@student.tuwien.ac.at

2 Basic concepts of fluid dynamics

This section covers the basics of systems that model fluid motion and is meant as an introduction in order to understand more sophisticated approaches.

2.1 The Navier-Stokes equations

The core component of a system that models fluid mechanics are the Navier-Stokes equations:

$$\frac{\partial u}{\partial t} = \underbrace{-(u\cdot\nabla)u}_{advection} - \underbrace{\frac{1}{\rho}\nabla p}_{pressure} + \underbrace{\nu\nabla^2 u}_{diffusion} + \underbrace{F_{ext}}_{external forces}, \quad (1)$$

$$\nabla\cdot u = 0, \quad (2)$$

where u denotes the velocity field, p stands for the pressure field, ν describes the material viscosity, ρ is the density and F_{ext} is the external force field. The left hand side of equation 1 depicts the temporal change of the velocity field and is called *momentum equation*. The Navier-Stokes system essentially describes how the motion of a particle in a fluid is continued. The meaning of the aforementioned symbols is kept consistent throughout the whole paper. Equation 2 is called the *incompressibility constraint* and necessary for providing stable solvers.

The momentum equation consists of four major terms that can be described intuitively.

Advection: The term advection refers to the inherent property of a fluid which makes objects inside it follow a certain flow. However the equation also describes the movement of the velocity field by itself which is modeled by the term $(u \cdot \nabla)$. As a result, not only objects are moved by the velocity field, it moves or advects itself. This is also the reason why solving fluid dynamics is hard. It is still not clear if there always exists an analytic smooth solution to these equations. This is called the *Navier–Stokes existence and smoothness* problem which is part of the *Millennium Prize Problems*. Figure 1 depicts the advection of smoke densities. In

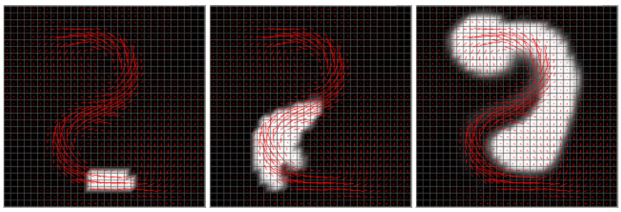


Figure 1: The advection moving a set of smoke particles along a static velocity field [Stam 2003].

this image, the velocities are not advected.

Pressure: Pressure forces that are exerted on each fluid are modeled by this term. The gradient of the pressure field ∇p points towards the steepest ascend of the field. However, due to the laws of physics, the particle is pushed from a higher pressure region to a lower one which is denoted by the minus sign. Since denser particles are harder to accelerate, the force is divided by the particles density ρ. An example for this behavior can be described by smoke particles that dissolve faster than thick liquid substances such as honey due to the lower amount of density present within smoke.

Diffusion: Different concentrations inside a fluid tend to average themselves out in order to achieve an equilibrium state. Dropping a viscous substance into a fluid results into diffusion. Depending on the viscosity, the diffusion proceeds more rapidly or slowly. This is the reason for the appearance of ν in the equation. The Laplace Operator ∇^2 models the diffusion in a sense that it has a high value if the average difference of the neighboring values of u is high. If the adjoint velocities differ to a smaller extent, the diffusion force is weaker. Therefore the optimal equilibrium state is reached if $\nabla^2 u = 0$.

External force field: The external force field allows to add additional forces which can model heat, user interaction, wind or user defined constraints such as shaping the fluid motion in a particular fashion that is suitable for the artists intentions. For example, an artist could define a target pressure model to which the fluid should converge. An illustration of such a user defined shaping can be seen in Figure 2. In some circumstances [Fedkiw et al. 2001] a smoke simulator can be defined that employs a temperature model that further alters the experienced velocity. The force term is called buoyancy and evaluates as follows:

$$F_{buoy} = \Big(-\alpha\rho + \beta(T - T_{amb}) \Big) z. \tag{3}$$

In this case, z points in the vertical direction and T_{amb} is the temperature of the air. Furthermore, $\alpha > 0$ and $\beta > 0$ are physically meaningful constants and T is the temperature of smoke. The first term $-\alpha\rho$ is a gravitational force which pulls heavier particles towards the ground. The second term $\beta(T - T_{amb})$ is positive when

Figure 2: Shaping fluids into a structure by using external force fields. The image was taken from the 2012 reel video of Fusion CI Studios.

$T > T_{amb}$ and acts as an uplifting force, otherwise it accelerates the gravitational pull.

In addition to equations 1 and 2, the movement of the density field ρ is often incorporated into fluid systems [Stam 2003]. The equation has a similar notion as the momentum equation:

$$\frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + \kappa \nabla^2 \rho + S. \tag{4}$$

The intention is to model the advection of the densities that is caused by the first term on the right hand side of equation 4. The diffusion is denoted by the second term and the term S corresponds to the addition of an external force specific to the density field ρ. Similar to the viscosity constant ν, κ denotes the strength of the densities to which extent they do withstand diffusion. The same principle of the momentum equation can also be applied to other scalar fields besides ρ such as the temperature T for instance.

The **incompressibility condition** states that the fluid conserves mass and can be understood intuitively. As we integrate the influx and efflux of a fluid over an arbitrarily shaped, closed surface with boundary $\partial\Omega$, the result should always yield zero. This is due to the fact that no density is lost inside the volume. Mathematically speaking, the equation is written as follows:

$$\int\int_{\partial\Omega} u \cdot n \, d\Omega = \int\int\int_{\Omega} u \cdot \nabla \, d\Omega = 0, \tag{5}$$

where n denotes the normal along the boundary $\partial\Omega$. In order to resolve to 0, the integrand $u \cdot \nabla$ must be 0 and this yields the incompressibility condition. A vector field u that fulfills this condition is also called *divergence free* [Stam 1999].

2.2 Enforcing the Incompressibility Condition

Upon solving equation 1 of the Navier-Stokes system, it needs to be ensured that the resulting vector field u is *divergence free*. Since a solver operates on time based iterations where each iteration covers a time interval Δt, this constraint needs to be enforced after each time step. This step is usually performed by solving a *Poisson equation* [Ando et al. 2013] or via the *Helmholtz-Hodge Decomposition* [Stam 1999]. The latter states that any vector field w can be decomposed into:

$$w = u + \nabla q, \tag{6}$$

where u is divergence free and q is a scalar field. Therefore, u can be made *divergence free* by applying the projection $\mathbf{P}$:

$$u = \mathbf{P}(w) = w - \nabla q. \tag{7}$$

In terms of the first Navier-Stokes equation this means that we need to solve:

$$\frac{\partial u}{\partial t} = \mathbf{P}\big(-(u\cdot\nabla)u + \nu\nabla^2 u + F_{ext}\big), \tag{8}$$

where the term $\mathbf{P}(\nabla p)$ vanishes due to Equation 7 and because of the linearity of $\mathbf{P}$. Furthermore, $\mathbf{P}\left(\frac{\partial u}{\partial t}\right) = \frac{\partial u}{\partial t}$ as $\mathbf{P}(u) = u$ because of the divergence free property of u and the same holds for the partial derivative as long as u is smooth.
As mentioned before, the second option to extract the divergence free part of any vector field is accomplished by solving a Poisson equation. If we multiply both sides in Equation 6 with ∇, we end up with:

$$\nabla w = \nabla^2 q, \tag{9}$$

as $\nabla \cdot v = 0$. This is precisely the Poisson equation. As Navier-Stokes equations are solved over discrete time steps this equation turns into:

$$\Delta t \nabla^2 q = \nabla w \tag{10}$$

In order to extract the divergence free part, we need to compute the minimal amount of energy that is necessary to change the state into an incompressible one. This can be achieved via energy minimization:

$$q = \arg\min_q \int_\Omega \frac{1}{2}\|w - \Delta t \nabla q\|^2 d\Omega. \tag{11}$$

2.3 Vorticity Confinement

Another important property of fluids is the concept of *vorticity confinement*. This aspect of fluids is not directly composed inside the Navier-Stokes equations but still corresponds to a component of every fluid motion. It becomes particularly important if the fluid solver is executed on a coarse grid which means that some amount of high frequency details cannot be properly computed. One way to add such lost energy into the system is by using the concept of vorticity confinement [Fedkiw et al. 2001]. This effect becomes apparent when fluids of low viscosity are rapidly accelerated by the velocity u. The flow might follow a turbulent pattern which can also be observed when the fluid is pushed around an obstacle as depicted in Figure 3. The vorticity of a vector field can be described by:

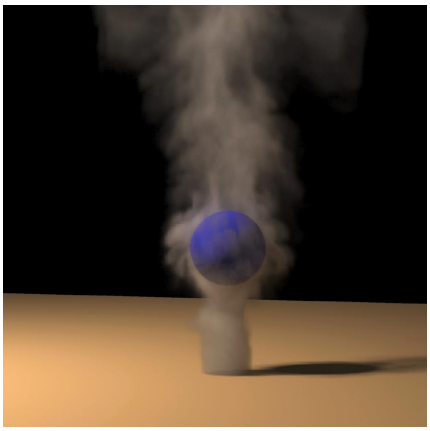


Figure 3: Smoke particles that are pushed around a ball where high vorticity can be observed [Fedkiw et al. 2001].

$$\omega = \nabla \times u, \tag{12}$$

which holds for incompressible flows. After obtaining the vorticity, a force can be defined that is directed towards the center of the corresponding vortex:

$$\mu = \nabla|\omega|. \tag{13}$$

The resulting force field is depicted in Figure 4. As μ conveys not

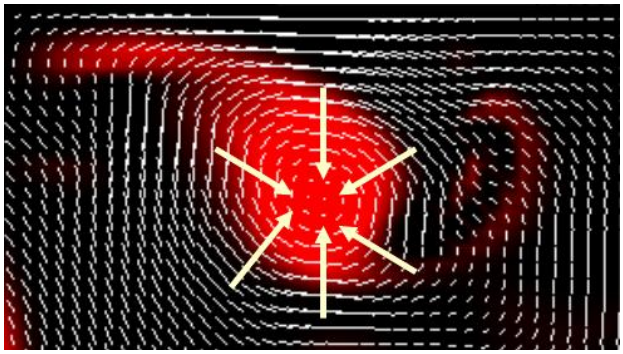


Figure 4: Visualization of the force field μ [Fedkiw et al. 2001].

only the direction, but also the size of the vortex, this term is normalized by a division through $|\mu|$ and called N. The reason is that the strength of the rotation in the vortex should be adjusted manually. The rotation is obtained by taking the cross product between N and ω multiplied by a weighting term ε:

$$F_{conf} = \varepsilon h(N \times \omega). \tag{14}$$

In Equation 14 the parameter h corresponds to the amount of spatial discretization. The higher the discretization factor is, the faster small scale details such as vorticity will be damped out. Therefore the added force $\varepsilon(N \times \omega)$ should be scaled appropriately.

2.4 Solving the Navier-Stokes on a Finite Eulerian Grid

A numerical solution to the Navier-Stokes equations is composed of the following steps that are described in [Fedkiw et al. 2001]. Initially, scalar, pressure and density fields are stored in memory together with a predefined force field F_{ext} or a model to obtain such external forces. A velocity field may or may not be present as it can evolve over time. The next steps require:

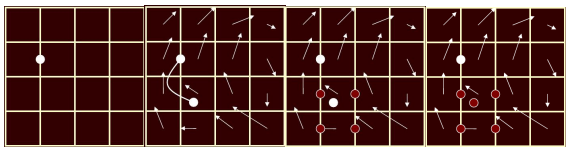
1. Computing and adding external forces F_{ext},
2. advecting the velocity field,
3. computing the diffusion and
4. applying the projection operator $\mathbf{P}$ described in equation 8.

The evaluation of these steps is usually performed on a 2 or 3 dimensional grid where finite differentials of the Nabla and Laplace operators can be used instead of continuous derivatives. Furthermore, those iterations are performed for a finite time interval Δt as mentioned in section 2.2. The coarseness of the grid in both space and time determines the numerical error of the method which is also dependent on the order of the solver that is used. Using a dense grid requires more time demanding computations but at the same time the accuracy of the solution is improved. Advanced methods described in further sections deal with this trade-off. A key component of the computation involves the evaluation of the advection term which should not be solved via a forward projection. The velocity vectors in these grids can reside at the center of each grid

cell or at the faces of each cell, depending on the used algorithm, different strategies are employed.
Supposing that the initial state of u is $u(x,t)$, where x is the position of the grid cell and t is the current time, we can evaluate the system as follows. The partial solution $u_1(x,t+\Delta t)$ is obtained after adding the forces:

$$u_1(x,t+\Delta t) = u(x,t) + \Delta t F_{ext}, \tag{15}$$

where the assumption has been made that the forces do not vary significantly during Δt.
Concerning the advection, a system of finite differences could be described and solved, also denoted as forward projection. This method however, is not stable in a sense that the velocity values will quickly blow up and diverge if the time step Δt is not sufficiently small. Dealing with small time steps is also infeasible because of the computational effort. The key idea of a stable solution can be understood intuitively and is presented in [Stam 1999] as well as [Stam 2003]. Instead of forward projecting velocities, a



(a) Particle (b) Tracing (c) Grid velocities (d) Interpolation

Figure 5: Tracing the velocities back in time [Stam 2003].

backtracing step is introduced. In the previously obtained velocity field $u_1(x,t+\Delta t)$, the point x should have been advected from t to $t+\Delta t$. Therefore this point is traced backwards along the path $p(x,s)$ on which it traveled from the previous time step to the current one. Its velocity is then changed to the velocity of the previous time step at the location $p(x,-\Delta t)$. The newly acquired force is then:

$$u_2(x,t+\Delta t) = u_1\big(p(x,-\Delta t), t+\Delta t\big) \tag{16}$$

At this point it is important to consider that the point $p(x,-\Delta t)$ must not lie on the center of a grid cell, actually, this is rarely the case. Therefore, its true velocity value must be obtained by interpolating the velocities on the neighboring grid cells or faces, depending on where the velocities are stored. Figure 5 illustrates the backtrace step along with the interpolation, in this case, the velocities are stored at the grid cells faces. In subfigure 5a, the velocity $u_1(x,t+\Delta t)$ is shown, backtracing is depicted in subfigure 5b. The neighboring velocities of the point at $p(x,-\Delta t)$ are shown in subfigure 5c. The final velocity is then interpolated with bilinear interpolation as the calculations are performed on a 2 dimensional grid. The interpolation can be changed to a cubic or trilinear model if more accuracy is needed or the grid size is 3 dimensional. This method of calculating the advection is also known as the *semi-Lagrangian scheme*.
As a next step, the diffusion needs to be computed that is equivalent to the standard wave equation:

$$\frac{\partial u_2}{\partial t} = \nu \nabla^2 u_2. \tag{17}$$

A solution to this equation can be computed by solving a discretized version of ∇^2.

Level-set functions: As explained in Section 2.1 any physical quantity such as densities, temperatures or velocities can be advected by u. In case of an Eulerian grid, it would be insufficient to model the whole n dimensional space because only a portion of all cells contain fluids. Therefore, a level set function $\varphi(x,t)$ is used to describe the subset of grid cells that contain the fluid. It is an implicit surface function where $\{(x,t) : \varphi(x,t) = 0\}$ describes the boundary of the fluid. The signed distance of $\varphi(x,t)$ is stored on each grid cell x and is either positive when it is outside of the liquid or negative when it lies inside of the liquid. The evolution of a liquid in space and time is described by:

$$\frac{\partial \varphi}{\partial t} = -u \cdot \nabla \varphi. \tag{18}$$

3 Frequency-Based Upsampling

When working on improvements of the numerical dissipation produced by finite and coarsely spaced simulation grids, information about the frequency spectrum that fluids convey can help in producing a more realistic output. This section explains how frequency analysis can be used in the fluid simulation. The goal is to generate artificial noise patterns and add them to the existing fluid in a realistic way using spectral components. The requirement of a realistic noise function is that it should contain high frequency details that are not present in the system. Furthermore, the noise should be added only at the positions where such information is lost.

3.1 Wavelet Noise and Wavelet Theory

A highly valuable tool in examining frequency information of input signals is the Fourier-Transform. This transformation precisely extracts the frequency components of a input function. However, one can not distinguish at which point in time certain frequencies exist in the signal. This is due to the *Gabor limit* which states that it is not possible to determine a functions power spectrum for both the frequency and time information with arbitrary fineness [Cohen 1995]. In case of the Fourier-transform, precise localization of the frequency is possible by completely sacrificing the time localization. In contrast, the wavelet transform allows time and frequency analysis in conjunction, however, the *Gabor limit* is still present. This is reflected by the fact that precise frequency analysis of lower frequency components is possible but the time localization is poor and vice versa for high frequency components.
Wavelet functions are important when well constrained noise patterns need to be constructed. The first approach in designing such procedural textures at different frequency levels was made in [Perlin and Velho 1995]. The authors explain how artificial details in a texture can be added to an image and its magnified and reduced versions. If an image is modified at the resolution $2^n \times 2^n$ and later changed to a resolution of $2^{n-1} \times 2^{n-1}$ it needs to be blurred before this change can be applied. The blurring is necessary, otherwise the picture at the new resolution would contain aliasing artifacts because of the high frequency components that cannot be properly displayed at a lower resolution. Filtering the image with a Gauss kernel is the typical approach when removal of such problematic high frequency components is desired. If the picture is again upsampled to $2^n \times 2^n$ these lost details need to be added back and this is done by storing them in a separate $2^n \times 2^n$ matrix. A *bandpass pyramid* can be created that contains the lost frequencies up to scale $2^1 \times 2^1$. [Perlin and Velho 1995] also describes that noise can be added and removed beyond the initial maximum resolution of an image. This resolution corresponds to the *Nyquist limit*. The method however, is not entirely aliasing free as discussed in [Cook and DeRose 2005]. Therefore, the authors of this work introduced a noise generation method that produces no aliasing artifacts even when noise patterns are created beyond the *Nyquist limit*. This is especially important if upsampled

fluid movements need to be created that lie above the resolution of the finite grid cells, e.g. above the *Nyquist limit* of the simulation.

The authors of [Cook and DeRose 2005] proved that there exists a noise function $N(x)$ that can be upsampled and downsampled without creating additional aliasing artifacts, even if the noise is above the *Nyquist limit* of the image:

$$\int N(2^j x - l)K(x-i)dx = 0. \tag{19}$$

The above equation describes the downsampling of random noise N generated at resolution $j \geq 0$ while the image has a resolution of $j = -1$. $K(x-i)$ is the smoothing kernel that is applied. Equation 19 evaluates to zero because the noise in higher resolutions is orthogonal to the filter kernel. Thus $N(x)$ has no effect on images with a lower resolution.
Suppose a basis function $\phi(x)$ was given at an arbitrary resolution which is called 0. We can build other functions from ϕ by using linear combinations of the input:

$$F(x) = \sum_i f_i \phi(x-i). \tag{20}$$

The set of all such functions with varying f_i is called the vector space S^0.Concerning noise patterns, it is important that noise on a higher resolution enriches the space S^0 such that $S^0 \subset S^1$. This behavior can be enforced by choosing *refinable* basis functions ϕ such as B-splines. If this criterion is not satisfied, noise patterns would inconsistently change during subsequent upsampling steps, resulting in visual artifacts.
Furthermore, all functions $G(x) \in S^1$ must be representable in S^0 without generating aliasing artifacts. This can be achieved by:

$$G(x) = G^{\downarrow}(x) + D(x), \tag{21}$$

where $G^{\downarrow}(x)$ is the approximation of $G(x)$ that lies in S^0 obtained with least squares error minimization. $D(x)$ contains the part of $G(x)$ that is unrepresentable in S^0. In order to prevent aliasing, $D(x)$ must be orthogonal to all functions in S^0. The vector space of all functions $D(x)$ that satisfy this constraint is called the wavelet space W^0. If the noise function resides in the aforementioned vector space, it won't affect images on lower resolutions.
Using this knowledge, noise patterns can be created as follows:

1. Generate random coefficients $R = (\ldots, r_i, \ldots)$.
2. Use B-spline basis functions $B(x)$ to create $R(x) \in S^1$.
3. $R(x) = \sum_i r_i B(2x-i)$.
4. Compute the least squares approximation $R^{\downarrow}(x)$.
5. Decompose $R(x) = R^{\downarrow}(x) + N(x)$ as shown in equation 21.
6. Extract $N(x) = \sum_i n_i B(2x-i)$.

3.2 Wavelet turbulence

The authors of [Kim et al. 2008] propose an upsampling technique for fluid systems that relies on the results explained in section 3.1. The core idea of the paper is to solve the Navier-Stokes on a coarse grid and subsequently add details to the simulation where they are lost. Furthermore, temporal coherence of the newly added frequencies is preserved and the results are synthesized into a grid of refined size. This allows an artist to precompute the Navier-Stokes and add fine details later during post-processing when they are needed. Since wavelets are used, high frequency details do not interfere with the low frequency components produced by the solver.

The wavelet noise function $N(x)$ is adjusted to model tur-

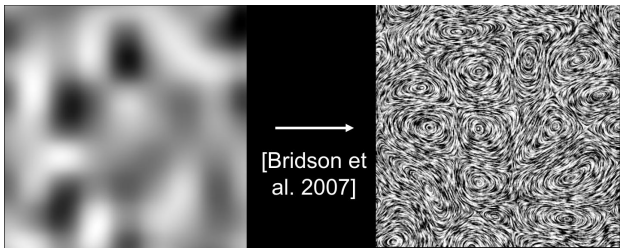


Figure 6: Wavelet noise $N(x)$ on the left is used to create a turbulent flow on the right $w(x) = \nabla \times N(x)$ [Kim et al. 2008].

bulent flows as depicted in Figure 6. The model is similar to the vorticity that was determined form the velocity u, however it can also be computed from the noise pattern itself leading to:

$$w(x) = \left(\frac{\partial N_1}{\partial y} - \frac{\partial N_2}{\partial z}, \frac{\partial N_3}{\partial z} - \frac{\partial N_1}{\partial x}, \frac{\partial N_2}{\partial x} - \frac{\partial N_3}{\partial y} \right). \tag{22}$$

It should be noted that $N(x)$ is designed as a scalar function, so the authors let N_1, N_2 and N_3 correspond to one single noise function with an offset for each N_*. Taking partial derivatives from the noise function does not destroy its band-limited properties as the derivative of a function in the frequency domain is linear. The partial derivatives are obtained by differentiating the weights of the B-spline function $B(x)$. Furthermore, the resulting field $w(x)$ retains its incompressibility.

A central concept is the property of *forward scattering* and *backward scattering* inherent to fluids. Whenever fluids are in motion, they generate large and small swirls that are denoted as *eddies*. When larger eddies move through the velocity field, some amount of compression and distortion leads to a break up into smaller eddies. This type of movement is called forward scattering and shown in Figure 7. Similarly, smaller eddies can be fused

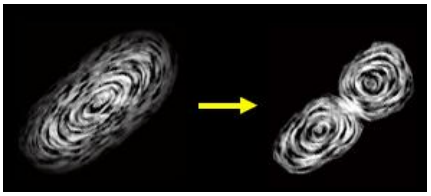


Figure 7

together yielding the backward scattering effect. It is desirable to detect such behavior and incorporate those fine scale effects when they would happen on a small grid simulation. This can be achieved with Kolmogorov's results in fluid analysis. A velocity field always conveys kinetic energy in each grid cell x:

$$e(x) = \frac{1}{2}|u(x)|^2. \tag{23}$$

Kolmogorov's analysis extracts the frequency components of the total energy e_t of the system at time t. The term e_t corresponds to the summation of every grid cells energy. As stated previously in section 3.1 the wavelet transform allows the retrieval of frequency and time information and is thus computed for various scales k on

the input function e_t. The result is Kolmogorov's five-thirds power law:

$$e_t(k) = C\varepsilon^{\frac{2}{3}}k^{-\frac{5}{3}}, \tag{24}$$

which states that within fine scale details, a decrease in kinetic energy following a slope of $-\frac{5}{3}$ steepness can be observed. Equation 24 can be computed recursively:

$$e_t(2k) = e_t(k)2^{-\frac{5}{3}}, e_t(1) = C\varepsilon^{\frac{2}{3}}. \tag{25}$$

Furthermore, 23 can be combined with 25 to create another recursion:

$$|\hat{u}(x,2k))| = |\hat{u}(x,k)|2^{-\frac{5}{6}}, |\hat{u}(x,1))| = 2^{\frac{1}{2}}C^{\frac{1}{2}}\varepsilon^{\frac{2}{6}}, \tag{26}$$

where $|\hat{u}(x,k)|$ is the spectral component of u constructed on the frequency band k. The final wavelet turbulence function can then be constructed:

$$y(x) = \sum_{i=i_{min}}^{i_{max}} w(2^i x)2^{-\frac{5}{6}(i-i_{min})}. \tag{27}$$

The deviation of equation 26 would suggest that we use $|\hat{u}(x,2^i)|$ instead of $w(2^i x)$, however since we do not have the high frequency information of u at certain bands above the *Nyquist limit* the noise function w that generates the additional detail must be substituted accordingly. In equation 27, i_{min} and i_{max} are used to constrain the noise creation for a certain range of frequency bands.

Grid Expansion: As the simulation runs on coarse grid n^3, but the turbulence is computed for a fine grid N^3 where $N^3 > n^3$, an appropriate injection step needs to be defined. Simply interpolating the coarse grid would result in a smoothed out version of the input velocities and does not generate new turbulence, therefore the authors suggest a model that computes the energy of the smallest eddie $e_t\left(\frac{n}{2}\right)$ and weights the turbulence function with this term. Additionally, turbulent eddies should only be added when forward scattering occurs, therefore the authors make use of the energy spectrum in $e_t\left(\frac{n}{2}\right)$ to detect such a process. If the high resolution grid N^3 is constructed, the density field can be advected on it. Except for the turbulence addition and successive advection, the computations are performed on the low resolution grid introducing only minor overhead into the simulation while achieving drastic visual improvements. In addition, the algorithm can be parallelized because the individual steps of the extrapolation and turbulence creation rely on local information on the grid. The authors use OpenMP to speed up the simulation and report a performance improve of 3.7 compared to a single core execution. Furthermore, the technique is independent of the underlying numerical solver.

Results of Wavelet Turbulence:

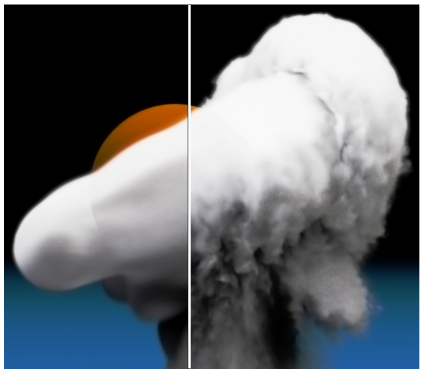


Figure 8: Comparison of the simulation on the low resolution grid 50^3 (left), and the high resolution synthesis on a 400^3 grid (right) [Kim et al. 2008].

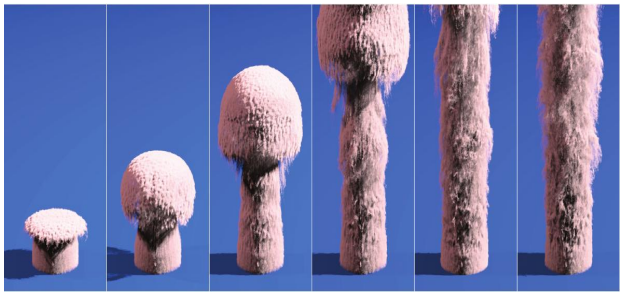


Figure 9: A wavelet turbulence simulation of smoke performed on a $50 \times 100 \times 50$ coarse grid, eight frequency bands where used and synthesized into a $12800 \times 25600 \times 12800$ grid. The simulation achieved a framerate of 170s on 8 cores [Kim et al. 2008].

4 Higher order advection solvers

Another possibility of improving visual results of the Navier-Stokes solver is to refine the underlying numerical method. Whereas in the previous section, artificial detail was added in a physically realistic fashion, the presented algorithms in this section deal with the numerical error that is produced by the finite resolution of an Eulerian grid. The key is to estimate the error of the advection step and correct it. The methods in this section do not only improve the flow locally as for example vorticity confinement, they also improve it on a macroscopic level since the overall motion is improved due to a smaller numerical error in the advection step.

4.1 BFECC

BFECC is a shorthand notation for a technique called the Back and Forth Error Correction and Compensation [Dupont and Liu 2003]. It is applied to the level set function and compensates the error in the semi-Lagrangian advection step. The algorithm is composed of the following steps:

1. Solve equation 18 using the semi-Lagrangian method and obtain $\tilde{\varphi}(x,t+\Delta t)$.
2. Solve equation 18 backward in time and denote the result as $\breve{\varphi}(x,t)$.
3. Compute the error e that is introduced by the forward and backward steps: $\breve{\varphi}(x,t) = \varphi(x,t) + 2e,\ e = -\frac{1}{2}(\varphi(x,t) - \breve{\varphi}(x,t))$.
4. Solve equation 18 again with $\bar{\varphi}(x,t) = \varphi(x,t) - e$ being the initial value.

The key is to incorporate the error e that is introduced by the Lagrangian method and compensate it by subtraction in the last step. The BFECC method can be applied to the advection of velocities, densities and level sets as described in [B. Kim 2005]. The paper outlines that, upon employing velocity advection with the BFECC method, physically realistic small scale fluctuations around obstacles on coarse Eulerian grids can be achieved. Furthermore, rigid body motion of objects in a fluid can be computed more accurate. This was shown by dropping a cup into water. The object sinks immediately into the water when the standard advection method is

applied, whereas the BFECC advection causes the cup to tumble correctly and subsequently it sinks to the ground. The price to pay is the additional invocation of a semi-Lagrangian solver, however the method is still computationally cheaper than refining the grid. In fact, the advection has a computational cost of $\Theta(n^3)$ and increases exponentially with n, whereas the increase in complexity with BFECC is linear: $\Theta(3n^3)$.

4.2 MacCormack Method

The MacCormack method which is described in [Selle et al. 2008], further reduces the computational complexity of the BFECC algorithm. Step 4 in the latter algorithm can be described by $A(\varphi(x,t)-e)$, where A denotes the semi-Lagrangian advection step. As A is linear and e is not a quantity that needs to be advected since it does not convey any information of φ, the formula can be written as $A(\varphi(x,t))+A(e)=A(\varphi(x,t))+e$. Because of the fact that we already computed $A(\varphi(x,t))=\bar{\varphi}(x,t+\Delta t)$ in step 1, we can omit the third advection step and only compute $\bar{\varphi}(x,t+\Delta t)-e$. This reduces the complexity to $\Theta(2n^3)$. The result of the algorithm in figure 10

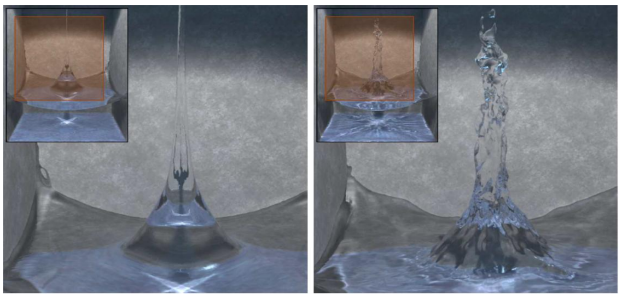


Figure 10: Results of the MacCormack advection step (right) in comparison with the standard, semi-Lagrangian approach [Selle et al. 2008].

emphasises the fact that much more details are preserved.

5 Spatio-Temporal Error Compensation

This method, which is described in [Zhang and Ma 2013] is again an error compensation technique. The key difference of this procedure in comparison to the algorithms in Section 4 is the solver independence. There is no explicit need to employ a linear semi-Lagrangian step. The computational overhead of the method in comparison to a fine grid simulation that achieves the same results is also kept small. Furthermore, it can be applied to other algorithms that operate on a mesh, instead of a 3 dimensional grid and can be incorporated into existing fluid solvers.

The algorithm uses a coarse grid in space and time, together with a small grid in order to identify the underlying error. This is the first discussed technique that also considers the error involved for larger Δt steps. When solving 1 on a grid of size Δx and Δt, one can prove that the error is of the following magnitude:

$$E(u,\Delta x,\Delta t)=E_x\Delta x^{k_x}+E_t\Delta t^{k_t}+O(\Delta x^{k_x+1},\Delta t^{k_t+1}). \quad (28)$$

In this equation, E_x and E_t depend on the solver and k_x together with k_t correspond to the order of the solver. As this error exists over the interval Δt it has to be multiplied by this quantity yielding $E\Delta t$, where the parameters of E have been omitted. Suppose that the coarse grid is $\hat{u}_0(m_x i\Delta x, m_t j\Delta t)$, where $m_x>1$ and $m_t\geq 1$ are integers. The error during the next time step $m_t(j+1)\Delta t$ can then be computed by substituting the aforementioned quantities into equation 28 and the result is:

$$u-\hat{u}_0=E_0 m_t\Delta t, \quad (29)$$

where u denotes the accurate solution and $\hat{u_0}$ is the numerical solution obtained from a coarse grid simulation. The same procedure can be applied to a fine grid with $m_x=1$, yielding $\hat{u_1}$. The resulting error is therefore:

$$u-\hat{u}_1=E_1 m_t\Delta t=E_x m_t\Delta x^{k_x}\Delta t+E_t m_t\Delta t^{k_t+1}+O(\Delta x^{k_x+1}\Delta t,\Delta t^{k_t+2}). \quad (30)$$

It is important that E_0 and E_1 are different since the grid spacing differs, therefore, E_0 is always bigger than E_1. Equations 29 and 30 can be combined into a single equation:

$$u-\frac{m_x^{k_x}\hat{u}_1-\hat{u}_0}{m_x^{k_x}-1}=E_2. \quad (31)$$

In E_2, there is no occurrence of Δx^{k_x} which means that the error term is even smaller than E_1, in addition, if $m_x^{k_x}=m_t^{k_t}$, the temporal error Δt^{k_t} also vanishes. This can be achieved by using the same solvers for both the fine and the coarse grids. So far the procedure looks quite promising but it can be improved further.
Concerning the fine grid, a numerical correction step can also be employed here. Until now, only points that lie on both grids have been considered and because m_x is an integer, the coarse grid always consists of a subset of fine grid points. The missing part is the approximation of the error term at a point that does only correspond to the fine grid. When starting at a fine grid point (i,j), the points where no coincidence with the coarse grid is present can be indexed by $(i+s,j)$ where $0<s<m_x$. At point (i,j) we have the following error:

$$u-\hat{u}_1=F_{i,j}+O(\Delta x^{k_x+1}\Delta t,\Delta t^{k_t+2}). \quad (32)$$

The missing error at the other points is $F_{i+s,j}$ and the authors showed that this error can be computed via interpolating the error at (i,j) and $(i+m_x,j)$. At this point in time, all points from the fine grid can be used for advection which will result in an improved numerical solution contrarily to simply computing the fine grid points.

Simulation Steps and Performance: The simulation is performed on the coarse grid and the small grid in parallel. The sizes were chosen such that the smaller grid has two times the temporal and spatial resolution of the bigger one. Because of this setup there need to be 2 fine grid simulation steps for one coarse step. Afterwards, the extrapolation and interpolation for error reduction is performed. Concerning the performance, the algorithm is only slightly slower than the standard simulation on the refined grid. Figure 11 shows the duration of a single iteration employed on both algorithms with varying grid sizes.

Implementation and Results: The authors of [Zhang and Ma 2013] implemented the Navier-Stokes solver in NVIDIA CUDA. The different extrapolation solvers on the grid are executed in parallel on the CPU. The simulation was run with different advection schemes such as semi-Lagrangian and BFECC. As mentioned earlier, the value of the parameter k_x decreases, with an increasing order of the solver. In general, extrapolation produces sharper density fields than normal simulation. Furthermore, different, high frequency fluid motion patterns are recovered when k_x is small enough, for example when BFECC is used in conjunction. The higher the value k_x, the closer the result will be compared to the standard simulation as can be observed in figure 12.
Another interesting result becomes apparent with the extrapolation

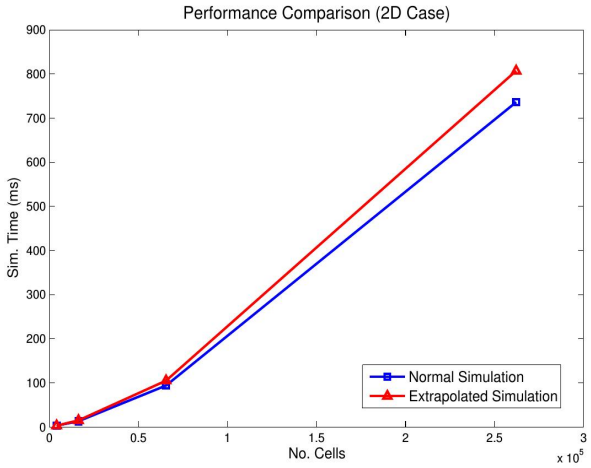


Figure 11: Performance evaluation of the Spatio-Temporal error correction (red) versus the standard Navier-Stokes semi-Lagrangian solver (blue) [Zhang and Ma 2013].

method as it produces more fine grained details on a 256×512 grid than a standard simulation on a 512×1024 grid which is illustrated in figure 13. Additionally, the authors compared the vorticity confinement algorithm with their algorithm, since extrapolation is capable of creating vortices in combination with BFECC. The resulting difference is that vorticity confinement produces stronger large scale vortices that are destroying the symmetry of the fluid, whereas the extrapolation focuses on the overall motion that is more consistently advected. This is due to the fact that the extrapolation predicts the fluid behavior more precisely and thereby improves the vorticity as well instead of just scaling it by a certain factor. Interestingly, as the proposed algorithm is solver-independent, it can be used in conjunction with a vorticity confinement-based solver.

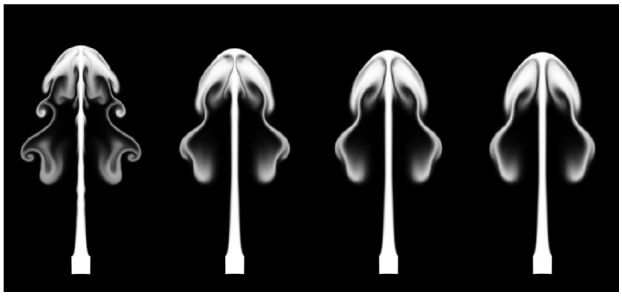


Figure 12: Comparison of different k_x values for the extrapolated simulation. From left to right $k_x = 1, k_x = 2, k_x = 3$ and no extrapolation [Zhang and Ma 2013].

6 Particle Based Liquid Simulation

So far, the Eulerian, grid-based approach of fluid simulation has been discussed to a large extent. There is however, another formalism of fluid dynamics, in particular, the Lagrangian one, where fluids are modeled as particles and the computations are performed on them without the use of Eulerian grid cells. The approach in [Mercier et al. 2015] deals with such a Lagrangian viewpoint

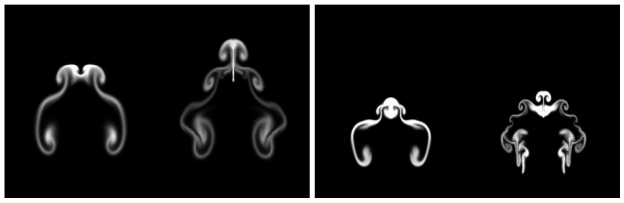


Figure 13: Comparison of the normal simulation (in each image on the left) and the extrapolated version (in each image on the right). The left image is computed on a 256×512 grid, the right image on a 512×1024 grid [Zhang and Ma 2013].

while at the same time borrowing concepts from Eulerian simulations. The goal of this work is to inject turbulence into a fluid system via wave propagation. Instead of focusing on frequency information or numerical error approximation, the features of the fluid surface are computed and analyzed in order to insert interesting motion where it is necessary.
The simulation is performed on a set of coarse sized particles before smaller particles are introduced in a fashion that preserves the coarse results but at the same time enriches the simulation by adding new dynamic structures using a wave equation. What's more, the method does not use a mesh that is refined, it constructs the surface out of the particles and at the same time borrows the concepts of level-sets that were used to generate meshes conveying the whole fluid.
The main steps of the algorithm that will be discussed thoroughly are as follows:

1. In the **surface maintenance** steps, the *advection* of fine scale points is computed,
2. followed by *surface normal* evaluation which,
3. is used to enforce *smoothness* constraints on the points.
4. In the **wave simulation** steps, the *curvature* of the surface is evaluated,
5. followed by a *wave seeding* strategy.

Surface maintenance: A special parameter λ_c is defined that allows the user some amount of control over the advection and curvature computation steps. For the sake of readability, coarse points positions at time n are denoted with X_i^n, while their refined counterparts are described with an x_i^n. The initial set of small particles x_i is created by uniformly sampling along spheres of radius λ_c centered at each X_i. During the advection, the value $2\lambda_c$ is used to restrict the amount of neighboring particles X_k^{n-1} that influence the advection of a certain small particle x_i^{n-1} to the position x_i^n. The advection of the smaller particles is carried out by averaging the positions of the coarse particles in the neighborhood $2\lambda_c$.
It is important to ensure that the fine scale particles do not drift away from their coarse counterparts, as the overall motion should be predictable. To keep the points in a certain range, a surface is constructed out of the set of coarse particles. Around each X_i two implicit spheres with corresponding centers and different radii r and R are defined where $R = \lambda_c, r = \frac{R}{2}$. Each point x_i stays within the defined ball that corresponds to its coarse neighbor. Using the union of all spheres as a complete surface for the fluid makes further steps like wave propagation difficult because of the discontinuities between neighboring spheres. Therefore, a smoothing constraint $g(f(y))$ is defined for a point y that ensures a continuous first derivative along the surface. Figure 14 describes how the constraint is reshaping the original surface.

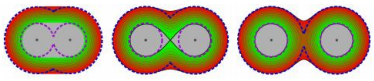


Figure 14: The defined surface within R and r (blue dots) is constrained to a continuous function (from green to red) [Mercier et al. 2015].

In order to simulate waves, the distribution of the points x_i must be improved. To achieve this goal, normals n_i for each x_i are computed via fitting a plane onto the smooth surface. Each particle is now displaced along its normal where the amount depends on the placement of its neighbors x_k in a radius λ_c. The idea behind this step is to reposition outliers such that no unusual bumps occur when the particles are fused to a common fluid surface. The point distribution is improved by moving each x_i away from their neighbors along $x_i's$ tangent plane. In addition, points are deleted if the density is too high, or inserted if there is not a sufficient amount of neighbors present.

Wave Simulation: With the previous steps a scenario is created that allows for additional turbulence propagation, the key step in this algorithm. The idea is that waves should always form in regions where many particles collide or separate. Such regions can be identified by computing the mean curvature for each point. As a tangent plane has been established previously, the value that can be compared to mean curvature is defined as the distance of $x_i's$ neighbors from the tangent plane of particle i and called c_i. A

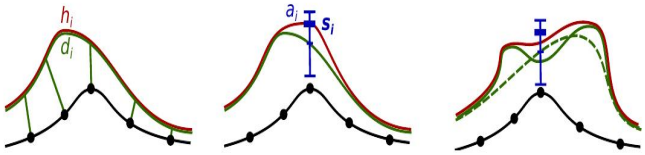


Figure 15: Wave seeding strategy on a high curvature region [Mercier et al. 2015].

wave traveling along the surface will be amplified if the value of c_i is high enough. The authors first compute an amplitude a_i that depends on the mean curvature and subsequently add up multiple cosine functions, multiplied by a_i, that have a certain frequency range provided by the user. The result of the addition is stored in s_i. The algorithm then solves a wave equation with the s_i values added to the height of the current wave d_i resulting in a new height h_i (Figure 15 middle in red). It has to be ensured that only frequencies are displayed that actually propagate out of the wave, therefore, h_i is never displayed. Instead, d_i is recomputed after solving the wave equation with h_i ,by subtracting s_i from the newly obtained h_i. In figure 15, this behavior is displayed. The left image shows the values h_i and d_i without any amplification. In the middle, s_i is shown after amplification together with h_i. The subtraction after solving the wave equation is shown in green on the right. The dashed green line symbolizes the wave propagation if no additional frequencies would have been added. As an additional step, the authors never use the Laplace-Beltrami operator to solve the wave equations, instead, they approximate it with a flat Laplacian that has proven to yield more stable results. Furthermore, the computational cost is reduced by this approximation.

Results and Performance: The authors compared their method to a standard particle solver that does not perform upsampling steps. Figure 16 shows a simulation with 12.5 million particles on the left. On top of this simulation the authors generate 500000 fine surface points and apply their wave evolution strategy. It is immediately visible that the resulting image on the right conveys more fine structured details where the original surface has a more smooth structure. It is remarkably that the simulation can still be improved even if it contains a large amount of particles.
In Figure 17 a scenario is shown where the flow follows a riverbed with obstacles in between. The standard simulation has 400000 particles and the upsampled version adds another 280000 small scaled surface points. The amount of additional detail is immediately visible.
In Figure 18 a high resolution solver (left) with 4 million particles is compared to the discussed algorithm operating on a lower resolution of 2500 coarse particles and 15500 small particles. The devised algorithm yields sharper waves and is also more efficient to compute since the high resolution simulation took 142s/frame in comparison to 4.74s/frame for the new method.
The authors use a hash structure for speeding up the lookups along the neighborhood region. With this, the simulation is 200 times faster on 290000 particles in comparison to iterating over all particles for finding neighbors. Furthermore, since most of the computations on the surface points are independent from each other, OpenMP is used to further accelerate the algorithm making the affected operations 8 times faster.

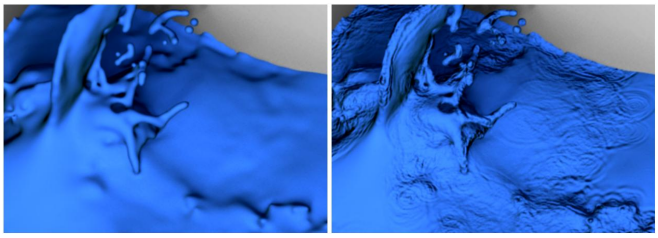


Figure 16: Wave seeding strategy on a high curvature region [Mercier et al. 2015].

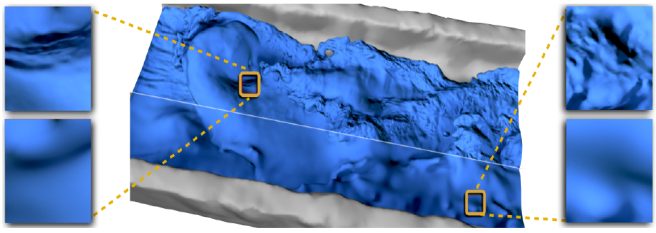


Figure 17: Wave seeding strategy on a high curvature region [Mercier et al. 2015].

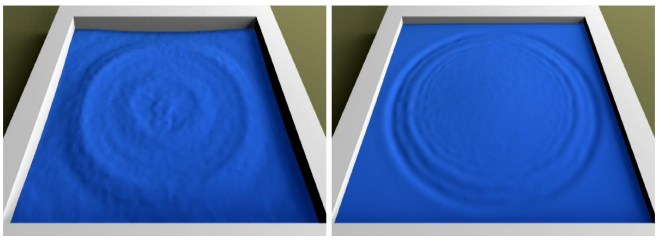


Figure 18: Wave seeding strategy on a high curvature region [Mercier et al. 2015].

7 Adaptive Simulation

The procedure in [Ando et al. 2013] tackles the upsampling problem in various ways. As discussed, using a uniform, Eulerian grid, is infeasible since it cannot focus on interesting motion and resolve it in more detail while at the same time wasting computational resources at places where the fluid motion is uninteresting and the calculations could be performed more coarsely. The authors therefore devised a solver which operates on a tetrahedral mesh. This structure is adaptively refined, depending on where interesting motion occurs. The algorithm is of hybrid nature, meaning that it still follows an Eulerian approach because the pressure and velocity is stored on the tetrahedral cells, but at the same time it uses particles from which a surface is computed. In the past, similar methods have been established using octrees, however, the authors in [Ando et al. 2013] denote that this approach still suffers from numerical errors and tetrahedral meshes are superior.

Fluid Solver: On each triangle of the mesh, a 3 dimensional velocity vector is stored at the cells center and a pressure value is stored on each vertex of the triangle. This setup allows the computation of the Poisson projection step modeled in equation 11 in an efficient and robust fashion. The energy minimization of the Poisson equation is modeled as a linear $m \times n$ system, where m is the number of tetrahedra and n is the number of nodes. As the number of nodes is always smaller than n, this system can be solved fast and without artifacts. Artifacts appear when more degrees of freedom for the pressure values are possible than for the velocities. By construction, this method prevents such a scenario as the authors claim, although a formal proof has not been given. Similar to [Mercier et al. 2015], the algorithm employs a position correction step where particles are moved away from its neighbors along the tangential direction of the surface. In order to prevent holes by using this correction, particles that lie underneath the surface will also be pushed towards the boundary of the surface, filling up empty space left behind. As an additional improvement, particles that move freely in space outside of the fluids boundary (for example, when splashes occur) will be excluded from the pressure solver and only gravitational forces are then applied to them as there are zero internal forces present in such cases.

Adaptivity: The adaptive component in the simulation is the tetrahedral mesh, which is recomputed every 10 time steps. The establishment of a tetrahedral mesh follows the algorithm from [Labelle and Shewchuk 2007]. During the recomputation of the mesh, all particles are inspected individually and it is determined whether they are either to big or too small. A particle is merged with a neighbor if it is too small. In the opposite case, a particle will be split and two new particles are formed in a way that tries to fill the gaps between their neighbors.
The crucial step is to determine when a split or merge operation needs to occur. To determine this, a special sizing function has been devised:

$$S(x) = max(d(x), V(x, min(\kappa_{liquid}(x), \kappa_{solid}(x), e(x)))). \quad (33)$$

In this equation, $d(x)$ is the distance of the particle from the surface of the wave. Motion near the surface must be modeled more precisely than motion farther below. $V(x,y)$ is a function that returns y if the point is inside the view frustum, otherwise, the result is the maximum allowed particle radius. The value y results form 3 different computations. The curvature is represented by $\kappa_{liquid}(x)$, the higher this value, the smaller the particle radius should be and vice versa. The distance to the nearest solid object is smoothly evaluated by $\kappa_{solid}(x)$ with respect to the maximum radius:

$$\kappa_{solid}(x) = 1.6\left(\frac{(1 - \|d_{solid}\|^2)}{r_{max}^2}\right)^3, \quad (34)$$

where d_{solid} is the distance to the solid object and r_{max} is the maximum radius. The idea is that particles near the solid object must be modeled more fine grained. The function $e(x)$ evaluates the strain tensor of the velocity field. This tensor describes the rate of change in velocities around a certain point and therefore conveys information about parts in the flow that convey interesting motion.

Surface Representation: The authors also propose a new method for representing the surface. This is necessary as previous methods can not handle many particles with such an ample difference in their radii. The final displayed surface is composed of the union of convex hulls that are formed by 3 neighboring particles that are close to the surface border. For the convex hull candidates, only particles that are lesser than l times the sum of the surface points and its neighbors radius apart are considered. A small l allows for the depiction of more details, that might be too bumpy and a large l smooths out concavities. The authors used $l = 2$ for most of their simulations, the convex hull of three particles is shown in Figure. After the surface has been established, the level set value from each vertex in the mesh is then computed. The resulting surface however, is expensive to compute with an average runtime of 5 minutes per frame.

Results and Performance: The most remarkable result of this algorithm can be analyzed in Figure 19, where a scene is depicted that would be difficult to compute with a standard solver. The fine details that occur within the splashes could otherwise only be achieved with a very fine grid. The authors note that a resolution of 400 million particles would have been needed to obtain the same accuracy as they achieved on the finest level of adaptivity. In comparison their method used 1.7 million particles, small scale details can be captured because of the sizing function. On average, the simulation took 4.6 seconds per frame, although it is presumed that the authors left out the time for the surface computation otherwise they would contradict themselves. In figure 20, another scene is shown that demonstrates the adaptivity of the method. The complex obstacle in the middle creates various amounts of distortion in the flow and the method is able to capture this by refining the particle size in regions around the obstacle. Regarding possible performance improvements, the authors state that the mesh generation also poses a performance bottleneck in their simulations as it can not parallelized. Further drawbacks are the configuration of the sizing function. There are constants that are not optimized perfectly and artifacts can occur in small areas when particle sizes vary too much. Furthermore, their method is not proven to behave correctly when viscosity and diffusion is modeled and the authors note that difficulties could arise in these cases. The repositioning of particles is also not entirely physically accurate but a necessity as a configuration could result that has more particles in a certain area than velocity samples in the grid. This would also lead to artifacts.

8 Comparison and Conclusion

In summary, all methods that have been discussed in detail try to improve fluid simulations on some level. The wavelet turbulence method, together with vorticity confinement add motion to a fluid in a procedural fashion that is not entirely physically accurate but

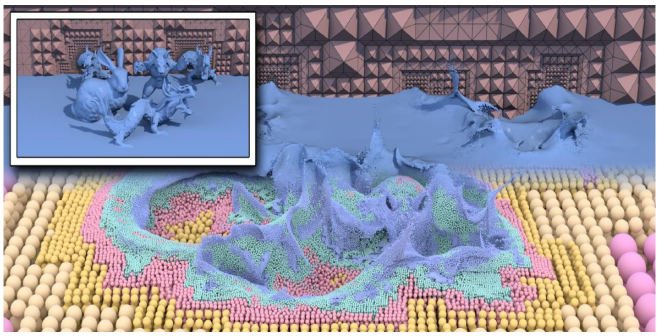


Figure 19: A scene that is difficult to evaluate precisely with non adaptive simulations because highly detailed splashes occur when the objects begin to flow and drop into the water [Ando et al. 2013].

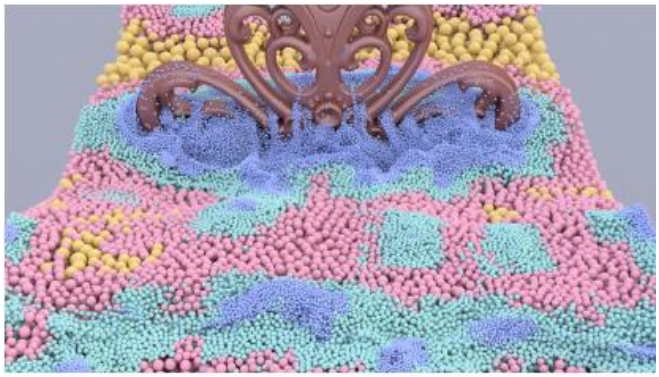


Figure 20: This scene depicts a obstacle with small an big holes that create small scale splashes and swirling motions when the water flows through it. Therefore an adaptive simulation that refines the resolution in this area is beneficial [Ando et al. 2013].

yields convincing visual results. In contrast, the higher order advection solvers together with the spatio-temporal method provide a purely numerical refinement that is closer to physical reality but the freedom of finetuning certain parameters may not be as pleasing. The hybrid approaches try to combine Eulerian and Lagrangian simulations and also apply some regularizations that have no real physical meaning, nevertheless they are also able to achieve a convincing amount of added small scale detail. The approaches all have their benefits and drawbacks, it is crucial to understand that different scenarios require different techniques. Each technique offers a huge amount of improvement compared to non-upsampling standard solvers, both in computational complexity as well as in the visual results that can be achieved.

8.1 Comparison of the Particle Based Solvers

Concerning the Particle-based solvers, the adaptive simulation might present superior simulations, while the Surface turbulence method might not be able to model fine details as accurate. When concerning the performance, the latter approach might be more convenient for artists. The authors mention that the framerate, even on complex scenes is always below 1 minute, this superiority in comparison with the adaptive simulation allows for a more rapid development process, which might be favored by artists. After a preferred scene has been established, one can choose to refine the simulation by employing the adaptive solver. It should also be noted that the adaptive version might not be able to simulate small waves in areas that are underresolved because the sizing function does not refine certain areas that are determined to convey little activity. When analyzing the right image in figure 18 small turbulent waves are simulated in areas that have only minor curvatures, the highly adaptive version might not be able to capture such details and is certainly not able to simulate additional waves.

8.2 Comparison of Particle Based solvers and Wavelet turbulence

Because of the wave seeding strategy the approach in [Mercier et al. 2015] can be compared to the wavelet turbulence algorithm [Kim et al. 2008]. Both methods inject turbulence at certain frequency bands, it is however not known how the simulation in [Mercier et al. 2015] behaves on scenarios where the motion of smoke is modeled. According to [Stam 2003], particle based methods are not able to capture the fine grained motions of smoke without using many particles. Because [Mercier et al. 2015] uses coarse particles as the underlying starting point for further refinement, the size of those particles might pose a restriction on the amount of fine grained turbulence in smoke simulations as for example in figure 9. Therefore, the method of choice when computing smoke simulations would be the wavelet turbulence approach, the higher order advection solvers or the spatio-temporal extrapolation method. It is not known whether the adaptive simulation on tetrahedral meshes is superior to the wavelet turbulence algorithm or not. The authors state that the applicability of the method with regards to smoke simulation is still a topic of further research. However, similar conclusions as mentioned above can be drawn. Although the particle size is refined in areas of interesting motion it is not clear if the adaption can correctly consider forward and backward scattering effects. It would be interesting to employ an adaptive wavelet turbulence sceme where the application refines a mesh based computation upon the detection of scattering effects. Furthermore, the wavelet turbulence method is not able to exactly reproduce simulations on very high resolution grids. Similar to the surface turbulence approach, the wavelet turbulence results are dependent on the low resolution simulations when obstacles are within the flow.

8.3 Comparison of Wavelet turbulence with numerical error correction based solvers

The wavelet turbulence method might allow more freedom when designing simulations, because the number of frequency bands that are added can be defined manually, depending on the needs of the artist. Simulations that focus solely on the improvement of numerical error terms such as BFECC, the MacCormack method and also the Spatio-Temporal extrapolation will always produce the same result. Still these approaches are highly valuable when there is need for physical accuracy since they do not add additional turbulence which is not entirely physically accurate. Such scenarios could be wind tunnel simulations for example. In addition, algorithms that focus on numerical improvements can be combined in order to achieve an even better result in a satisfactorily amount of time as discussed in [Zhang and Ma 2013].

References

ANDO, R., THÜREY, N., AND WOJTAN, C. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (TOG) 32*, 4, 103.

B. KIM, Y. LIU, I. L. J. R. 2005. Flowfixer: Using bfecc for fluid simulation. In *Proceedings of the Eurographics Work shop on Natural Phenomena*.

COHEN, L. 1995. *Time-frequency Analysis: Theory and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

COOK, R. L., AND DEROSE, T. 2005. Wavelet noise. *ACM Trans. Graph. 24*, 3 (July), 803–811.

DUPONT, T. F., AND LIU, Y. 2003. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *Journal of Computational Physics 190*, 1, 311–324.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 15–22.

KIM, T., THÜREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. In *ACM Transactions on Graphics (TOG)*, vol. 27, ACM, 50.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics 26*, 3 (July), 57.1–57.10. Special issue on Proceedings of SIGGRAPH 2007.

MERCIER, O., BEAUCHEMIN, C., THUEREY, N., KIM, T., AND NOWROUZEZAHRAI, D. 2015. Surface turbulence for particle-based liquid simulations. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2015) 34*, 6 (Nov.).

PERLIN, K., AND VELHO, L. 1995. Live paint: Painting with procedural multiscale textures. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 153–160.

SELLE, A., FEDKIW, R., KIM, B., LIU, Y., AND ROSSIGNAC, J. 2008. An unconditionally stable maccormack method. *Journal of Scientific Computing 35*, 2-3, 350–371.

STAM, J. 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 121–128.

STAM, J. 2003. Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, vol. 18, 25.

ZHANG, Y., AND MA, K.-L. 2013. Spatio-temporal extrapolation for fluid animation. *ACM Transactions on Graphics (TOG) 32*, 6, 183.

ZSOLNAI, K., AND SZIRMAY-KALOS, L. Real-time simulation and control of newtonian fluids using the navier-stokes equations.