

# Data-driven Fluid Simulations using Regression Forests

Lubor Ladický<sup>\*†</sup>

ETH Zurich

SoHyeon Jeong<sup>\*†</sup>

ETH Zurich

Barbara Solenthaler<sup>†</sup>

ETH Zurich

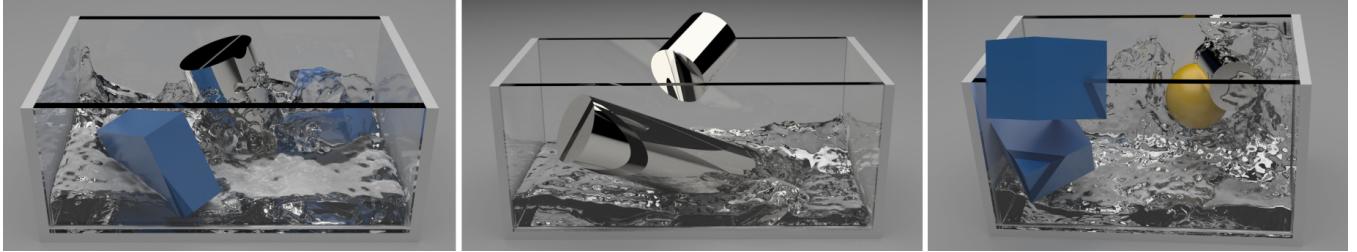
Marc Pollefeys<sup>†</sup>

ETH Zurich

Markus Gross<sup>†</sup>

ETH Zurich

Disney Research Zurich



**Figure 1:** The obtained results using our regression forest method, capable of simulating millions of particles in realtime. Our promising results suggest the applicability of machine learning techniques to physics-based simulations in time-critical settings, where running time matters more than the physical exactness.

## Abstract

Traditional fluid simulations require large computational resources even for an average sized scene with the main bottleneck being a very small time step size, required to guarantee the stability of the solution. Despite a large progress in parallel computing and efficient algorithms for pressure computation in the recent years, real-time fluid simulations have been possible only under very restricted conditions. In this paper we propose a novel machine learning based approach, that formulates physics-based fluid simulation as a regression problem, estimating the acceleration of every particle for each frame. We designed a feature vector, directly modelling individual forces and constraints from the Navier-Stokes equations, giving the method strong generalization properties to reliably predict positions and velocities of particles in a large time step setting on yet unseen test videos. We used a regression forest to approximate the behaviour of particles observed in the large training set of simulations obtained using a traditional solver. Our GPU implementation led to a speed-up of one to three orders of magnitude compared to the state-of-the-art position-based fluid solver and runs in real-time for systems with up to 2 million particles.

**CR Categories:** I.3.3 [Computer Graphics]: Computational Geometry and Object Modeling Physically based modeling; I.2.6 [Artificial Intelligence]: Learning;

**Keywords:** fluid simulation, data-driven, regression forest

## 1 Introduction

Computing high-resolution fluid simulations with traditional state-of-the-art approaches is very challenging, as they require tremendous computational resources to compute a scene with millions of particles. The main bottleneck is the severe restriction on the time step size needed to guarantee stability, and thus simulation times are typically in the range of hours to days on high-end computers, making it impossible to achieve high-resolution fluids in real-time.

The standard Smoothed Particle Hydrodynamics [Lucy 1977] (SPH) method approximates continuous quantities in the Navier-Stokes differential equations using discrete particles with appropriate smoothing kernel and replaces a continuous advection by an advection of particles. The approach does not deal with the incompressibility constraint directly, which causes significant visually unpleasant artifacts.

Recent work has addressed this problem and either enforce a density invariance condition or a divergence-free velocity field. A predictive-corrective scheme has been introduced where pressure values are iteratively updated to satisfy the zero compression constraint [Solenthaler and Pajarola 2009]. The performance has been further improved by discretizing and iteratively solving the pressure Poisson equation [Ihmsen et al. 2013]. Most recently, a position based fluid (PBF) approach has been presented [Macklin and Mueller 2013], where first all particles are advected, and then projected to the manifold of feasible solutions by iteratively correcting positions of particles to satisfy the incompressibility constraint. PBF allows to use a larger time step compared to its counterparts. A density invariance condition has also been combined with a multi-scale scheme [Horvath and Solenthaler 2013] to further speed-up the computation. Despite all these improvements, high-resolution fluids are still computed offline.

An alternative to particle-based approaches are grid-based methods that approximate continuous quantities on a discrete regular grid [Enright et al. 2002]. Incompressibility is enforced on the grid by solving the Poisson's equation, making the velocity field divergence-free. To speed up grid-based simulations, the solution space can be restricted to simpler topology [Chentanez and Müller 2010; Chentanez and Müller 2011]. To avoid discretization artifacts of grid-based methods, the hybrid FLIP model [Zhu and Bridson

<sup>\*</sup>The authors assert equal contribution and joint first authorship

<sup>†</sup>e-mail:lubor.ladicky,sohyeon.jeong,solenthaler,marc.pollefeys,grossm@inf.ethz.ch

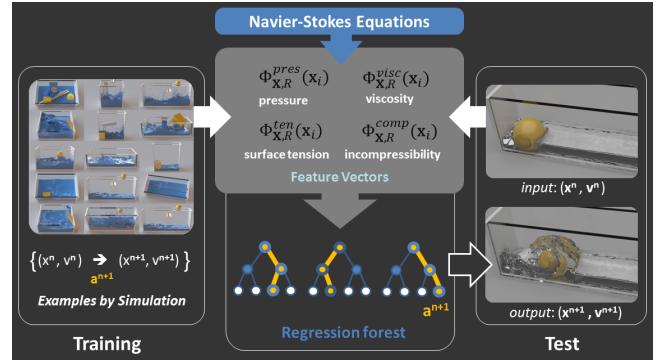
2005] solves advection on particles directly.

In order to address the problem of computational performance, various types of data-driven simulations have been explored. These methods do not depend on the discretization level, but rather use a reduced representation of the simulation space. Hence, they are targeted at computing interactive simulations that still are able to obtain fine details. The most common data-driven approaches for fluid simulations are based on Galerkin projection [Treuille et al. 2006; Gupta and Narasimhan 2007; De Witt et al. 2012], that transforms the dynamics of the fluid simulation to operations on linear combinations of preprocessed snap-shots (tiles). Reduced representations are typically obtained using dimensionality reduction techniques, such as Principal Component Analysis, restricting the richness of the representation, but providing the desired speed-up. The method had been extended to very large simulation spaces [Wicke et al. 2009] by decomposing the problem into simulation tiles with the dynamics operating only on the low dimensional reduced space. The consistency between tiles is enforced using a constraint reduction method. Cubature models exactly evaluate the fluid dynamics at a sparse set of domain positions and then extrapolate these values to the remaining area [Kim and Delaney 2013]. A state graph has been presented in [Stanton et al. 2014], building upon the observation that in simple games only a small set of fluid states are visited. At game play, the state graph is traversed and the best matching fluid state is displayed. Another data-driven approach [Raveendran et al. 2014] aimed to generate a large number of fluid simulations by interpolating existing preprocessed simulations.

In this paper we explore a novel data-driven fluid approach (see Figure 2, that is capable of simulating a large number of particles at interactive frame rates. Our machine learning method formulated as a regression problem is capable of learning and reliably predicting positions and velocities of particles in a setting with a very large time step. We designed a feature vector that fully represents the state of a particle and its context in a form that is rich enough to encode all necessary information required to predict the next state, and takes constant-time to evaluate. The individual dimensions of the feature vector directly correspond to individual forces and incompressibility constraint of the Navier-Stokes equations in surrounding regions at different scales. Using this feature vector we tested three different prediction and correction learning schemes to mimic various fluid simulation solvers. A large number of random training videos were generated using the position-based fluid method and used for training as a ground truth. As a regression method we used regression forests [Breiman 2001], a powerful approach successfully applied to various real-time tasks such as Kinect body-part regression [Taylor et al. 2012] or camera relocalization [Shotton et al. 2013]. Our method managed to successfully learn the behaviour of fluid particles from the training examples and provides a good alternative for real-time applications of fluid simulations such as computer games or interactive design. GPU implementation of our algorithm significantly outperforms any existing approaches in terms of running time by a factor of  $10 - 1000$  depending on the simulation parameters.

## 2 Designing the feature vector

In this section we describe the general form of the context-based feature vector we are going to use to model fluid dynamics. The restriction to fast regression methods, such as regression forest, yields several requirements for the form of a feature vector. It needs to model the behaviour of other particles in both close or far neighbourhood without the necessity to explicitly calculate nearest neighbours. The evaluation of a particular dimension of a feature vector should be possible in constant time, resulting in a running time linear with the number of particles. A feature vector needs to



**Figure 2:** A general concept of machine learning approach applied to fluid dynamics. Given a set of training videos, we extract training pairs of transitions between current and next state of each particle, build a feature vector, capable of modelling quantities in Navier-Stokes equations, and train a regressor, predicting the next state of each particle. During test stage for a new previously unseen scene we extract the feature vector for each particle and apply the trained regression model to predict the velocity of a particle as a correction to an advected state.

be rich enough to model all possible constellations of neighbouring particles. Furthermore, it should be robust to small deviations; a small  $\epsilon$ -change of the input should result into a small change of a feature vector. One such representation is based on context-based integral features, defined as a flat-kernel sums of rectangular regions surrounding the particle. In our case they will model "regional" forces and constraints of the Navier-Stokes equations. Their main advantage is the constant-time evaluation and robustness to a small shift (unlike point-wise features), crucial for systems with varying density of particles. Various forms of integral features have been previously used for example in the face detector of [Viola and Jones 2004] to calculate Haar wavelets, or in 2D semantic pixel-wise classifiers [Shotton et al. 2006; Ladicky et al. 2009] for the fast bag-of-words representation calculation.

### 2.1 Context-based integral features

Let  $S$  be a discretized space of points  $\mathbf{x} = (x, y, z) \in S$  and  $R$  be the box  $[x_{min}^R, x_{max}^R] \times [y_{min}^R, y_{max}^R] \times [z_{min}^R, z_{max}^R]$ , where  $x_{min}^R \leq x_{max}^R$ ,  $y_{min}^R \leq y_{max}^R$  and  $z_{min}^R \leq z_{max}^R$ , and let  $\Omega_R$  be the membership function for the box  $R$ :

$$\Omega_R(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in R \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The integral feature  $\phi_R^f$  of a point-wise feature  $f(\mathbf{x})$  is defined as sum of  $f(\mathbf{x})$  over the box  $R$ :

$$\phi^f(R) = \sum_{\mathbf{x} \in S} f(\mathbf{x}) \Omega_R(\mathbf{x}). \quad (2)$$

The integral feature is directly tied to the average response over the box; given  $R$ , the average response can be obtained from the integral feature just by simple division by the number of points in the box. The main strength of these features is the efficiency of their evaluation. In the preprocessing stage we calculate integral volumes, defined as:

$$\psi^f(x, y, z) := \phi^f([0, x] \times [0, y] \times [0, z]) \quad (3)$$

for each feature  $f$ . An integral volume can be calculated once for all particles in linear time with the size of the space  $S$ . For any box  $R$  an integral feature  $\phi^f(R)$  can be calculated in constant time as:

$$\begin{aligned}\phi^f(R) = & \psi^f(x_{max}^R, y_{max}^R, z_{max}^R) - \psi^f(x_{min}^R, y_{max}^R, z_{max}^R) \\ & - \psi^f(x_{max}^R, y_{min}^R, z_{max}^R) - \psi^f(x_{max}^R, y_{max}^R, z_{min}^R) \\ & + \psi^f(x_{min}^R, y_{min}^R, z_{max}^R) + \psi^f(x_{min}^R, y_{max}^R, z_{min}^R) \\ & + \psi^f(x_{max}^R, y_{min}^R, z_{min}^R) - \psi^f(x_{min}^R, y_{min}^R, z_{min}^R).\end{aligned}\quad (4)$$

The membership functions  $\Omega_{\bar{R}_k}$  over all boxes starting at zero  $\bar{R}_k = [0, x^{\bar{R}_k}] \times [0, y^{\bar{R}_k}] \times [0, z^{\bar{R}_k}]$  form a basis for any function  $f$ :

$$f(\mathbf{x}) = \sum_{\bar{R}_k} \alpha_{\bar{R}_k}^f \Omega_{\bar{R}_k}(\mathbf{x}) \quad (5)$$

with coefficients:

$$\begin{aligned}\alpha_{\bar{R}_k}^f = & \frac{1}{8} \left( \phi^f(x^{\bar{R}_k}, y^{\bar{R}_k}, z^{\bar{R}_k}) - \phi^f(x^{\bar{R}_k} + 1, y^{\bar{R}_k}, z^{\bar{R}_k}) \right. \\ & - \phi^f(x^{\bar{R}_k}, y^{\bar{R}_k} + 1, z^{\bar{R}_k}) - \phi^f(x^{\bar{R}_k}, y^{\bar{R}_k}, z^{\bar{R}_k} + 1) \\ & + \phi^f(x^{\bar{R}_k} + 1, y^{\bar{R}_k} + 1, z^{\bar{R}_k}) + \phi^f(x^{\bar{R}_k} + 1, y^{\bar{R}_k}, z^{\bar{R}_k} + 1) \\ & \left. + \phi^f(x^{\bar{R}_k}, y^{\bar{R}_k} + 1, z^{\bar{R}_k} + 1) - \phi^f(x^{\bar{R}_k} + 1, y^{\bar{R}_k} + 1, z^{\bar{R}_k} + 1) \right),\end{aligned}\quad (6)$$

implying that there is a bijection between all function values  $f(\mathbf{x})$  and integral features  $\psi^f(\mathbf{x})$ .

Let us consider the learning problem for a translation invariant quantity  $q(\mathbf{x}_i)$  as a function of its contextual features  $f(\mathbf{x}_k - \mathbf{x}_i)$ :

$$q(\mathbf{x}_i) := q(f(\mathbf{x}_1 - \mathbf{x}_i), f(\mathbf{x}_2 - \mathbf{x}_i), \dots). \quad (7)$$

The bijection between  $f(\mathbf{x})$  and  $\psi^f(\mathbf{x})$  means that without losing any information we can reformulate the learning problem as:

$$q(\mathbf{x}_i) := q(\psi^f(\bar{R}_1 + \mathbf{x}_i), \psi^f(\bar{R}_2 + \mathbf{x}_i), \dots), \quad (8)$$

or use more general over-representation with all boxes  $R_k$  (not only  $\bar{R}$  starting from zero) as:

$$q(\mathbf{x}_i) := q(\phi^f(R_1 + \mathbf{x}_i), \phi^f(R_2 + \mathbf{x}_i), \dots). \quad (9)$$

For a fluid simulation the learnt quantity  $q$  could be either acceleration or correction to advected velocity, as described in Section 3.

## 2.2 Integral features for SPH

We are interested in building a regressor predicting the position and the velocity of a particle in the next frame as a function of the positions and velocities of all particles in the current frame. As a feature vector, we could use the integral velocities and densities over regions placed relatively to the position of a particle. However, such a learning approach is doomed to fail. In the regression forest setting such features are not sufficiently discriminative to reach a conclusion about every expected outcome in a reasonable depth of a tree. The problem lies in the lack of invariance of such features. Context-based integral features are invariant to translation, however integral velocities are not inertia-independent. Ignoring this important symmetry would require a huge amount of training data, modelling all possible interactions at all possible absolute velocities of interacting particles, not only at all their relative velocities. Such symmetries of the fluid equations (or physics in general) could be enforced directly by building a feature vector using forces and constraints from the Navier-Stokes equations.

In SPH [Monaghan 1992], the state of physical quantities at any point of space is represented as an interpolation of the states of discrete particles using appropriate smoothing kernel functions. Any scalar quantity  $A(\mathbf{x})$  at a point  $\mathbf{x}$  is approximated using the set of particles  $\mathbf{X}$  as:

$$A(\mathbf{x}) = \sum_{j \in \mathbf{X}} \frac{m_j A_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j) = \sum_{j \in \mathbf{X}} \tilde{A}_j W(\mathbf{x} - \mathbf{x}_j), \quad (10)$$

where  $m_j$  is the weight,  $\rho_j$  is the density and  $A_j$  is the scalar quantity of the  $j$ -th particle,  $\tilde{A}_j = \frac{m_j A_j}{\rho_j}$ , and  $W(\mathbf{x})$  is a spherically symmetric smoothing kernel. This interpolation procedure is applied to the density and pressure fields, and in each iteration particle positions and velocities are integrated using the estimated accelerations by applying a standard Euler gradient step.

Our goal is to find the acceleration field by solving the Navier-Stokes equations for incompressible fluids:

$$\frac{d\mathbf{v}}{dt} + \mathbf{v} \cdot \nabla \mathbf{v} = \frac{1}{\rho} (-\nabla p + \mu \nabla^2 \mathbf{v} + \sigma \nabla^2 \mathbf{x}) + \mathbf{g}, \quad (11)$$

subject to the incompressibility constraint inside the fluid  $\rho(\mathbf{x}) = \rho_0$ .

The viscosity term in SPH is approximated as in [Müller et al. 2003]:

$$a_i^{visc} = \frac{\mu}{\rho_0} \sum_j (\mathbf{v}_j - \mathbf{v}_i) \nabla^2 W(\mathbf{x}_j - \mathbf{x}_i). \quad (12)$$

The corresponding integral viscosity feature (2) for any  $\nabla^2 W(\cdot)$  takes the form:

$$\begin{aligned}\Phi_{\mathbf{X}, R}^{visc}(\mathbf{x}_i) = & \frac{\mu}{\rho_0} \sum_{j \in \mathbf{X}} (\mathbf{v}_j - \mathbf{v}_i) \Omega_R(\mathbf{x}_j - \mathbf{x}_i) \\ = & \frac{\mu}{\rho_0} \left( \sum_{j \in \mathbf{X}} \mathbf{v}_j \Omega_R(\mathbf{x}_j - \mathbf{x}_i) - \mathbf{v}_i \sum_{j \in \mathbf{X}} \Omega_R(\mathbf{x}_j - \mathbf{x}_i) \right)\end{aligned}\quad (13)$$

The functions  $\Phi_{\mathbf{X}, R}^{visc}(\mathbf{x}_i)$  for all boxes  $R$  are invariant to inertia, modelling the acceleration due to the viscosity in the Navier-Stokes equations. The feature for a particular box  $R$  can be calculated efficiently using 4 integral volumes, one for density  $\psi_{\mathbf{X}}^N$  counting the number of particles  $N$  in a box, and three for each dimension of the velocity  $\psi_{\mathbf{X}}^{v_x}$ ,  $\psi_{\mathbf{X}}^{v_y}$ , and  $\psi_{\mathbf{X}}^{v_z}$ . The three components of the feature  $\Phi_{\mathbf{X}, R}^{visc}$  are obtained as:

$$\begin{aligned}(\Phi_{\mathbf{X}, R}^{visc})^x(\mathbf{x}_i) &= \frac{\mu}{\rho_0} (\phi_{\mathbf{X}, R}^{v_x}(\mathbf{x}_i) - v_i^x \phi_{\mathbf{X}, R}^N(\mathbf{x}_i)) \\ (\Phi_{\mathbf{X}, R}^{visc})^y(\mathbf{x}_i) &= \frac{\mu}{\rho_0} (\phi_{\mathbf{X}, R}^{v_y}(\mathbf{x}_i) - v_i^y \phi_{\mathbf{X}, R}^N(\mathbf{x}_i)) \\ (\Phi_{\mathbf{X}, R}^{visc})^z(\mathbf{x}_i) &= \frac{\mu}{\rho_0} (\phi_{\mathbf{X}, R}^{v_z}(\mathbf{x}_i) - v_i^z \phi_{\mathbf{X}, R}^N(\mathbf{x}_i)).\end{aligned}\quad (14)$$

The surface tension term, modelled using the Laplacian, can be approximated as [Thürey et al. 2010]:

$$a_i^{ten} = \frac{\sigma}{\rho_0} \sum_j (\mathbf{x}_j - \mathbf{x}_i) \nabla^2 W(\mathbf{x}_j - \mathbf{x}_i). \quad (15)$$

The corresponding integral tension feature (2) for any  $\nabla^2 W(\cdot)$  takes the form:

$$\begin{aligned}\Phi_{\mathbf{X}, R}^{ten}(\mathbf{x}_i) = & \frac{\sigma}{\rho_0} \sum_{j \in \mathbf{X}} (\mathbf{x}_j - \mathbf{x}_i) \Omega_R(\mathbf{x}_j - \mathbf{x}_i) \\ = & \frac{\sigma}{\rho_0} \left( \sum_{j \in \mathbf{X}} \mathbf{x}_j \Omega_R(\mathbf{x}_j - \mathbf{x}_i) - \mathbf{x}_i \sum_{j \in \mathbf{X}} \Omega_R(\mathbf{x}_j - \mathbf{x}_i) \right)\end{aligned}\quad (16)$$

The feature for a particular box  $R$  can be calculated similarly to viscosity using 4 integral volumes, for density  $\psi_{\mathbf{X}}^N$  and three for each dimension  $\psi_{\mathbf{X}}^x$ ,  $\psi_{\mathbf{X}}^y$ , and  $\psi_{\mathbf{X}}^z$ . The three components of the feature  $\Phi_{i,R}^{ten}$  are obtained as:

$$\begin{aligned} (\Phi_{\mathbf{X},R}^{ten})^x(\mathbf{x}_i) &= \frac{\sigma}{\rho_0} (\phi_{\mathbf{X},R}^x(\mathbf{x}_i) - x_i \phi_{\mathbf{X},R}^N(\mathbf{x}_i)) \\ (\Phi_{\mathbf{X},R}^{ten})^y(\mathbf{x}_i) &= \frac{\sigma}{\rho_0} (\phi_{\mathbf{X},R}^y(\mathbf{x}_i) - y_i \phi_{\mathbf{X},R}^N(\mathbf{x}_i)) \\ (\Phi_{\mathbf{X},R}^{ten})^z(\mathbf{x}_i) &= \frac{\sigma}{\rho_0} (\phi_{\mathbf{X},R}^z(\mathbf{x}_i) - z_i \phi_{\mathbf{X},R}^N(\mathbf{x}_i)). \end{aligned} \quad (17)$$

The pressure  $p$  for an incompressible fluid can be computed using the ideal gas state equation and is a linear function of the density [Desbrun and Cani 1996]  $p = k\rho$ , where  $k$  is a constant. Applying SPH to the pressure term we get:

$$a_i^{pres} = -\frac{k}{\rho_0} \nabla \sum_j m_j W(\mathbf{x}_j - \mathbf{x}_i). \quad (18)$$

The corresponding integral pressure feature (2) for any  $W(\cdot)$  can be defined using a discrete derivative as:

$$\begin{aligned} \Phi_{\mathbf{X},R}^{pres}(\mathbf{x}_i) &= \frac{k}{\rho_0} \nabla \sum_{j \in \mathbf{X}} \Omega_R(\mathbf{x}_j - \mathbf{x}_i) \\ &= \frac{k}{\rho_0} \sum_{j \in \mathbf{X}} \Omega_R(\mathbf{x}_j - \mathbf{x}_i + \Delta e^n) - \Omega_R(\mathbf{x}_j - \mathbf{x}_i - \Delta e^n), \end{aligned} \quad (19)$$

where  $\Delta e^n$  is the unit vector in the  $n$ -th dimension.

To model the pressure also from obstacles, we calculate the integral volume of the density  $\psi_{\mathbf{X}}^O$ , which includes not only fluid particles, but also particles composing the obstacles. Thus, the three components of the pressure feature  $\Phi_{i,R}^{pres}$  are calculated as:

$$\begin{aligned} (\Phi_{\mathbf{X},R}^{pres})^x(\mathbf{x}_i) &= \frac{k}{\rho_0} (\phi_{\mathbf{X},R}^O(\mathbf{x} + \Delta e^x) - \phi_{\mathbf{X},R}^O(\mathbf{x} - \Delta e^x)) \\ (\Phi_{\mathbf{X},R}^{pres})^y(\mathbf{x}_i) &= \frac{k}{\rho_0} (\phi_{\mathbf{X},R}^O(\mathbf{x} + \Delta e^y) - \phi_{\mathbf{X},R}^O(\mathbf{x} - \Delta e^y)) \\ (\Phi_{\mathbf{X},R}^{pres})^z(\mathbf{x}_i) &= \frac{k}{\rho_0} (\phi_{\mathbf{X},R}^O(\mathbf{x} + \Delta e^z) - \phi_{\mathbf{X},R}^O(\mathbf{x} - \Delta e^z)). \end{aligned} \quad (20)$$

The incompressibility constraint can be incorporated using the additional integral feature, modelling the restriction of the number of particles in a certain region as:

$$\Phi_{\mathbf{X},R}^{comp}(\mathbf{x}_i) = \phi_{\mathbf{X},R}^O(\mathbf{x}_i). \quad (21)$$

The final feature vector is a concatenation of the feature vectors of each component of the Navier-Stokes equations, calculated on a large fixed set of randomly sampled boxes  $R$ . It models the state of each particle without any loss of information, except the loss due to the discretization, which was reduced by applying trilinear interpolation when approximating a certain quantity of a particle into a grid. Each dimension of the feature vector can be interpreted either as a *regional* force or a constraint over a region.

### 3 Learning fluid dynamics

In the learning stage, our goal is to build a regressor, capable of predicting the next state of a particle  $i$  in each frame  $n$  using the feature vector  $\Phi_{\mathbf{X}}(\mathbf{x}_i)$ . The exact formulation of the regression problem has a very large impact on the final performance of the method. We considered three different ways to predict the next state, each one trying to mimic certain fluid dynamics solver.

**1. Learning naïve prediction** We can formulate the regression problem as:

$$\mathbf{a}_i^n := Reg(\Phi_{\mathbf{X}}(i)), \quad (22)$$

where  $\mathbf{a}_i^n$  is the acceleration of the  $i$ -th particle in the  $n$ -th frame and  $Reg(\cdot)$  is a learnt regression function. Using this prediction, we integrate the velocity and the position of each particle using midpoint integration as:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{a}_i^n \Delta t \quad (23)$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \frac{\mathbf{v}_i^n + \mathbf{v}_i^{n+1}}{2} \Delta t \quad (24)$$

where  $\Delta t$  is the time step between two frames. Numerical integration of this form models the remaining advection term in the Navier-Stokes equations. This formulation is able to directly learn the acceleration including gravity. This formulation does not directly deal with the incompressibility constraint, because the knowledge of densities is insufficient to predict where the fluid might get compressed. Conceptually, this approach tries to mimic the standard SPH method, where the acceleration is also directly predicted based on the current state of a system.

**2. Learning correction** Given a current state of the system, we first advect all particles using external force accelerations  $\mathbf{a}_{ext}$  (such as gravity) as:

$$\mathbf{v}_i^{n*} = \mathbf{v}_i^n + \mathbf{a}_{ext} \Delta t \quad (25)$$

$$\mathbf{x}_i^{n*} = \mathbf{x}_i^n + \frac{\mathbf{v}_i^n + \mathbf{v}_i^{n*}}{2} \Delta t, \quad (26)$$

apply the collision detection to  $\mathbf{x}_i^{n*}$ , and formulate the regression problem as learning of the correction:

$$\Delta \mathbf{v}_i^{corr} := Reg(\Phi_{\mathbf{X}^*}(i)). \quad (27)$$

The predicted correction is then applied as:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^{n*} + \Delta \mathbf{v}_i^{corr} \quad (28)$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^{n*} + \frac{\Delta \mathbf{v}_i^{corr}}{2} \Delta t. \quad (29)$$

For a fixed time step  $\Delta t$  the learning of the correction of velocity is equivalent to the learning of the correction of position. This approach directly handles incompressibility, because it can detect any possible compression after the advection and can do an appropriate adjustment. Unlike the naïve prediction it is self-correcting; any wrong correction in one iteration can be corrected in the next frame. Conceptually, the approach tries to mimic position based fluids (PBF) [Macklin and Mueller 2013], which corrects the positions of particles after advection, as well as grid based methods, which correct velocities after advection, such that the velocity field becomes divergent free. Unlike in PBF, the regressor takes into account a larger neighbourhood system, which does not require several iterations to converge.

**3. Learning prediction with hindsight** Given a current state of the system, we first advect all particles the same way as in the previous correction approach using (25) and (26) and apply collision detection. Then, we formulate the regression as:

$$\mathbf{a}_i^n := Reg(\Phi_{\mathbf{X}^*}(i)) \quad (30)$$

and do the advection as in (23) and (24). Unlike in the naïve prediction approach, the regressor is capable to predict possible fluid compression and to do a suitable adjustment of an acceleration. Conceptually, this approach is similar to the predictive-corrective incompressible SPH (PCISPH) approach [Solenthaler and Pajarola 2010].

---

**Algorithm 1** Regression tree evaluation.

---

**Input:**  $D, \Phi(\mathbf{x}), tree, outputDim$   
**Output:**  $\mathbf{q}$

```

for all  $i \in D$  do
     $depth = 0$ 
     $N = tree.root$ 
    while  $N^{term} = 0$  do
         $f_{depth} = \Phi(\mathbf{x})_{N^f}(i)$ 
        if  $f_{depth} \geq N^\theta$  then
             $N = N^{right}$ 
        else
             $N = N^{left}$ 
         $depth = depth + 1$ 
    for  $d = 1$  to  $outputDim$  do
         $q_d(i) = N^{c_d} \cdot \mathbf{f} + N^{b_d}$ 

```

---

2009], where accelerations are iteratively predicted including updated pressure forces.

These three proposed regression approaches can be in principle combined, however it would lead to a significant slow-down of the method.

### 3.1 Regression forests

A regression forest is an ensemble of uncorrelated binary regression trees, each one independently predicting a desired quantity  $\mathbf{q}$ . The final prediction of a regression forest is obtained by averaging predictions of all trees.

**Evaluation of a regression tree** An evaluation for a tree for a particle  $i$  is performed by starting in the root node  $N = tree^{root}$  and eventually reaching a leaf node by iteratively moving either to its right or left child (denoted as  $N^{right}$  and  $N^{left}$ ) based on the value of the decision function  $\delta(\Phi(\mathbf{x})_{N^f}(i) \geq N^\theta)$ , where  $N^\theta$  is the threshold and  $\Phi(\mathbf{x})(i)$  is the value of one particular dimension  $N^f$  of the feature vector  $\Phi(\mathbf{x})$  for a particle  $i$ . Note that not all feature dimensions of a feature vector have to be calculated for each particle to reach the leaf node, it can be at most as many as the maximum depth of a tree. The prediction of a given tree is the value of the output function  $\mathbf{q}^N(\cdot)$  of a reached terminal leaf node  $N$ . The most common output function is a constant vector  $\mathbf{q}^N(\Phi(\mathbf{x})) = \mathbf{q}_0^N$ . More complicated functions such as a linear function of a possibly high-dimensional input vector are typically not used to avoid over-fitting. Except the constant output function  $\mathbf{q}_0^N$ , we propose to use a good trade-off between constant and linear cases; the linear function of the feature used along the way between the root and a leaf node. Let  $\mathbf{N}_{path}$  be the set of internal nodes between a leaf node  $N$  and the root node, and  $\mathbf{f}^{\mathbf{N}_{path}}(\Phi(\mathbf{x}))$  the set of feature values, evaluated along the way. The linear output function of the leaf node  $N$  for each output dimension  $d$  takes the form:

$$q_d^N(\Phi(\mathbf{x})) = N^{c_d} \cdot \mathbf{f}^{\mathbf{N}_{path}(N)}(\Phi(\mathbf{x})) + N^{b_d}, \quad (31)$$

where  $N^{c_d}$  is a set of linear weights of the length limited by the maximum depth of a tree and  $N^{b_d}$  is the bias term. The use of this output function has several advantages. On the one hand, it is much more general than a constant function, on the other hand, with the sufficient amount of data per leaf node there is not much space for over-fitting. The increase of evaluation time is negligible for the learning problems where the slowest part is the on-fly evaluation of features, such as in our case. A detailed implementation of the regression tree evaluation is shown in the pseudo-code Algorithm 1.

---

**Algorithm 2** The first stage of regression tree training.

---

**Input:**  $D, featureDim, randomCount, outputDim, maxDepth, minSize$   
**Output:**  $tree$

```

tree.root = CreateNode()
AddToQueue(queue, [tree.root  D  0])
while NotEmpty(queue) do
     $[N \hat{D} depth] = GetFirst(queue)$ 
    if ( $depth < maxDepth$ ) & ( $|\hat{D}| \geq minSize$ ) then
         $N^{term} = 0$ 
         $err_{best} = \infty$ 
        for  $m = 1$  to  $randomCount$  do
             $feat = GetRandomNumber(1, featureDim)$ 
            for all  $\theta$  do
                 $\hat{D}^{right} = \{i \in \hat{D} | \Phi_k(i) \geq \theta\}$ 
                 $\hat{D}^{left} = \hat{D} \setminus \hat{D}^{right}$ 
                 $err = \frac{|\hat{D}^{right}|}{|\hat{D}|} var_{\mathbf{q}}(\hat{D}^{right}) + \frac{|\hat{D}^{left}|}{|\hat{D}|} var_{\mathbf{q}}(\hat{D}^{left})$ 
                if  $err < err_{best}$  then
                     $N^f = feat$ 
                     $err_{best} = err$ 
                     $N^\theta = \theta$ 
                 $\hat{D}^{right} = \{i \in \hat{D} | \Phi(\mathbf{x})_{N^f}(i) \geq N^\theta\}$ 
                 $N^{right} = CreateNode()$ 
                AddToQueue(queue, [ $N^{right} \hat{D}^{right} depth + 1$ ])
                 $\hat{D}^{left} = \hat{D} \setminus \hat{D}^{right}$ 
                 $N^{left} = CreateNode()$ 
                AddToQueue(queue, [ $N^{left} \hat{D}^{left} depth + 1$ ])
            else
                 $N^{term} = 1$ 

```

---

**Training of a regression tree** Training consists of two stages; learning the structure of each tree with the most discriminative decision functions (feature indexes  $N^f$  and thresholds  $N^\theta$ ) for each internal node  $N$ , and finding the output functions (coefficients  $N^{c_d}$  and a bias  $N^{b_d}$ ) for each leaf node  $N$  and output dimension  $d$ . The first stage is done in a standard greedy fashion as in [Breiman 2001]. Starting with the root node, for each unprocessed internal node in the queue  $N$  and its subset of data  $\hat{D}$ , that reaches this internal node, we find the most discriminative feature index  $N^f$  and a threshold  $N^\theta$  by minimizing the error (weighted variance):

$$err = \frac{|\hat{D}^{right}|}{|\hat{D}|} var_{\mathbf{q}}(\hat{D}^{right}) + \frac{|\hat{D}^{left}|}{|\hat{D}|} var_{\mathbf{q}}(\hat{D}^{left}), \quad (32)$$

where  $\hat{D}^{right}$  is the subset of the data  $\hat{D}$  that ends up on the right side of the tree:

$$\hat{D}^{right} = \{i \in \hat{D} | \Phi_{N^f}(i) \geq N^\theta\}, \quad (33)$$

$\hat{D}^{left} = \hat{D} \setminus \hat{D}^{right}$  is its complement,  $|\hat{D}|$  is the number of elements and  $var_{\mathbf{q}}(\hat{D})$  is the variance of the quantity  $\mathbf{q}$  on the set  $\hat{D}$ . The optimization of the error (32) is performed by randomly sampling a subset of dimensions of the feature vector  $\Phi(\mathbf{x})$ , brute force evaluation of the error for a set of thresholds  $\theta$  and remembering the best splitting function so far. The randomness in the training procedure guarantees that the individual trees will significantly differ from each other, making predictions more robust. Trees are stopped being grown when the number of samples is below a threshold or the depth of a tree reached its predefined limit. The learning of the structure of a tree is described in detail in the pseudo-code Algorithm 2.

The bottleneck of the first stage of training is typically a high memory consumption; the method needs to keep in memory either fea-

---

**Algorithm 3** The second stage of regression tree training.

---

**Input:**  $\Phi(\mathbf{x})$ ,  $outputDim$ ,  $tree$ ,  $D$

**for all terminal  $N$  do**

$$\mathbf{A}^N = \mathbf{0}, \mathbf{B}^N = \mathbf{0}$$

**for all  $i \in D$  do**

$$depth = 0$$

$$N = tree.root$$

**while**  $N^{term} = 0$  **do**

$$f_{depth} = \Phi(\mathbf{x})_{Nf}(i)$$

**if**  $f_{depth} \geq N^\theta$  **then**

$$N = N^{right}$$

**else**

$$N = N^{left}$$

$$depth = depth + 1$$

$$\mathbf{A}^N = \mathbf{A}^N + \begin{bmatrix} \mathbf{f} \\ 1 \end{bmatrix} \begin{bmatrix} \mathbf{f}^T & 1 \end{bmatrix}, \mathbf{B}^N = \mathbf{B}^N + \begin{bmatrix} \mathbf{f} \\ 1 \end{bmatrix} \mathbf{q}(i)^T$$

**for all terminal  $N, d \in \{1, \dots, outputDim\}$  do**

$$\begin{bmatrix} N^{c_d} \\ N^{b_d} \end{bmatrix} = (\mathbf{A}^N)^{-1} \mathbf{B}^N_d$$


---

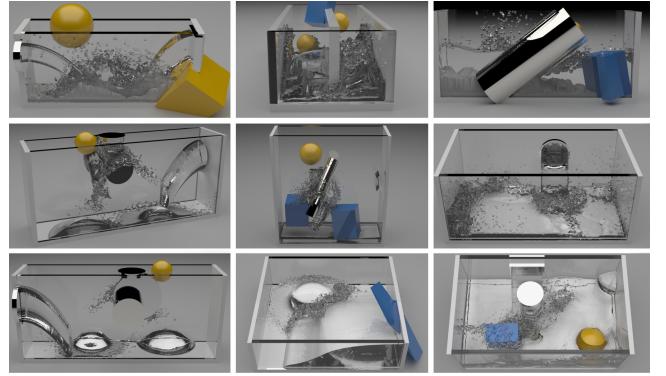
ture vectors of all the training data points or structures able to generate them (integral volumes in our case). Even using a large cluster of computers, the first stage learning process can very unlikely process more than one billion data points. We want to fit linear functions to each leaf node of a tree, and thus there would be not enough data to avoid over-fitting. To deal with this problem, we fix the structure of the trees using the result of the first stage process, and show that it is possible to deal with an almost arbitrary amount of data to determine optimal linear coefficients  $N^{c_d}$  and a bias  $N^{b_d}$  for each leaf node  $N$  and each output dimension  $d$  using a standard least-squares linear regression. Given a leaf node data  $\hat{D}$  and its set of features along the path  $\mathbf{f} := \mathbf{f}^{N_{path}}$ , the solution for each dimension  $d$  can be found in closed form as:

$$\begin{bmatrix} N^{c_d} \\ N^{b_d} \end{bmatrix} = \left( \sum_{i \in \hat{D}} \begin{bmatrix} \mathbf{f}(i) \\ 1 \end{bmatrix} \begin{bmatrix} \mathbf{f}(i)^T & 1 \end{bmatrix} \right)^{-1} \left( \sum_{i \in \hat{D}} \begin{bmatrix} \mathbf{f}(i) q_d(i) \\ q_d(i) \end{bmatrix} \right), \quad (34)$$

where  $[\cdot] [\cdot \cdot]$  is the outer product. Apparently, the solution can be found by just one pass through the data, keeping in memory all sums in a matrix and a vector from (34) without any need for keeping any information about each individual sample. For each new training sample we find the corresponding leaf node and update its sums. Thus, there are no memory restrictions as in the first stage of the learning and we can pass through all available data. If we have a generator of an arbitrary amount of data (such as obtained by a fluid solver for randomly generated scenes), we can essentially run this stage of learning till each leaf node contains a sufficient amount of data required to avoid over-fitting. In practice, we can process several orders of magnitude more data than in the first stage. The concept of fixing the structure of the tree and retraining only the leaf nodes can also be used to estimate the constant output function  $\mathbf{q}_0 = \frac{\sum_{i \in \hat{D}} \mathbf{q}(i)}{|\hat{D}|}$ . The learning of the linear coefficients of all leaf nodes of a tree is described in detail in the pseudo-code Algorithm 3. In general, training leaf nodes after fixing the tree can be used for any other regression or classification forest problems where a data generator is available (for example the artificial depth data generator for human poses in Kinect [Shotton et al. 2011]).

## 4 Experiments

**Training data acquisition** The training data was obtained using the PBF algorithm [Macklin and Mueller 2013] evaluated on the



**Figure 3:** Examples of the training data, obtained using PBF solver. For a small time step, PBF can generate a sufficiently high quality data at a reasonable computational cost.

set of 165 randomly generated scenes. Each scene consisted of a few bodies of water, sources and randomly moving, interacting, appearing and disappearing obstacles, such as spheres, cylinders, boxes or height map terrains. Each training video of 6 seconds (stored as 188 frames with 32ms time step) contained between 1 and 6 million particles of the radius 0.01m, yielding the requirement for a small simulation time step of 1 ms. 5 projections to a feasible manifold have been made in each 1ms time step. Even though the running time of our GPU implementation of the PBF algorithm under the same parameters was the same as in the original paper [Macklin and Mueller 2013], under our setting the total simulation time was approximately 50 minutes per video. Example snap shots from the training set are shown in Figure 3. For a comparison we also simulated 10 videos (a subset of the 165 PBF videos) using the PCISPH solver [Solenthaler and Pajarola 2009], taking approximately 10 hours per video to simulate. Comparisons of the results revealed that PBF might not have been the most suitable training data acquisition method for our approach. The data analysis showed that when interacting with obstacles, many particles require relatively large correction with respect to the time step to avoid compression, often corresponding to acceleration exceeding  $10g$  even for a not-moving stable body of fluid. Such problems were not observed with the PCISPH. To partially resolve the problem with the PBF training data, we weighted the influence of each training sample  $i$  by  $w_i = \exp(-\frac{\|\mathbf{a}_i\|}{2g})$ .

**Regression forest training** The first stage of training was done on 4% of the particles from a randomly selected subset of 12 randomly rotated and/or mirrored frames from each training video. This resulted in a total number of 100 million training samples. A random set of 5000 rectangular boxes of the Gaussian distribution have been sampled as potential candidates for features used in the regression trees. The mean and the variance of the Gaussian distribution has been determined based on a quick test, which uniformly distributed boxes tend to be chosen if only a small amount of data is used. The space was discretized to 0.02m, the same as the particle diameter, corresponding to approximately one particle per cell for an uncompressed fluid. The maximum depth of a tree was set to 20. The training time of the first stage was 4 days per tree on the cluster of 10 computers. The second stage of training was done on all particles from 300 randomly rotated and mirrored frames from each video with a total number of 60 billion training samples ( $600 \times$  more than in the first stage). Originally we intended to use the random scene generator of PBF simulations [Macklin and Mueller 2013] directly in the training pipeline without any necessity to store the intermediate data, however due to a rare divergence

Method	CPU runtime	GPU runtime
PBF	70.2s	5.25s
PCISPH	720s	100s
Regression forest	2.20s	24.08ms

**Table 1:** Comparison of approximate running times per frame ( $T = 32$  ms) of CPU and GPU versions of PBF, PCISPH and our regression forest for one million particles. CPU code was run on a single core 3.5GHz Intel processor, GPU version on NVidia GTX 780. We also implemented the multi-core CPU version (speeding up the method approximately by the number of cores). The performance is shown for a single core to get a fair comparison with other methods. PBF parameters (timestep, the number of projections) were the same as we used to build the training set. PBF was often stable also for a larger time step or smaller number of projections, but in rare cases it diverged. However, it was not possible to decrease its running time by an order of magnitude in our setting due to a small particle radius.

(approximately 5% of the scenes) of the random simulations this approach was not possible. The training time for the second stage was 8 hours per tree on the same cluster. Over 99% of the nodes containing in total over 99.9% training data had at least 5000 training samples, sufficient to avoid over-fitting. We trained both linear and constant output functions using this data. Even though the linear fit decreased the training and evaluation error (see the next paragraph), visually we did not observe much difference. Due to a large size of trained model (520MB vs 40MB) we did not use the linear fit for our real-time GPU simulations.

**Comparison of learning approaches** We trained one tree for each learning setup (prediction, correction, prediction with hindsight). The naïve approach was not able to deal with moving obstacles, which often led to a compression of the fluid. The algorithm was not able to self-correct in the next frames, because it has never seen such distorted states during training. The visual comparison between correction and prediction with hindsight determined the correction approach to be a clear winner, mainly due to more realistic splashes and better handling of the incompressibility constraint. To confirm our visual observation, we measured the absolute  $L^2$ -error of the predicted accelerations, re-initialized in every frame to the PBF solution. Relative errors are not suitable for comparison, because accelerations could be arbitrary close to 0 for a stable fluid. Surprisingly, the quantitative evaluation on the 30 new PBF test videos showed, that the prediction methods performed quantitatively better than the correction approach. The average error of naïve prediction was  $0.2066ms^{-2}$  ( $0.2150ms^{-2}$  with a constant function), the error of prediction with hindsight was  $0.2019ms^{-2}$  ( $0.2071ms^{-2}$  with a constant function), but the error of correction was  $0.2313ms^{-2}$  ( $0.2377ms^{-2}$  with a constant function). The absolute values should not be taken too seriously due to the already mentioned problem with PBF data. Quantitative comparisons in a longer run would be misleading due to the instability of the differential equations with respect to the  $\epsilon$ -change of initial conditions.

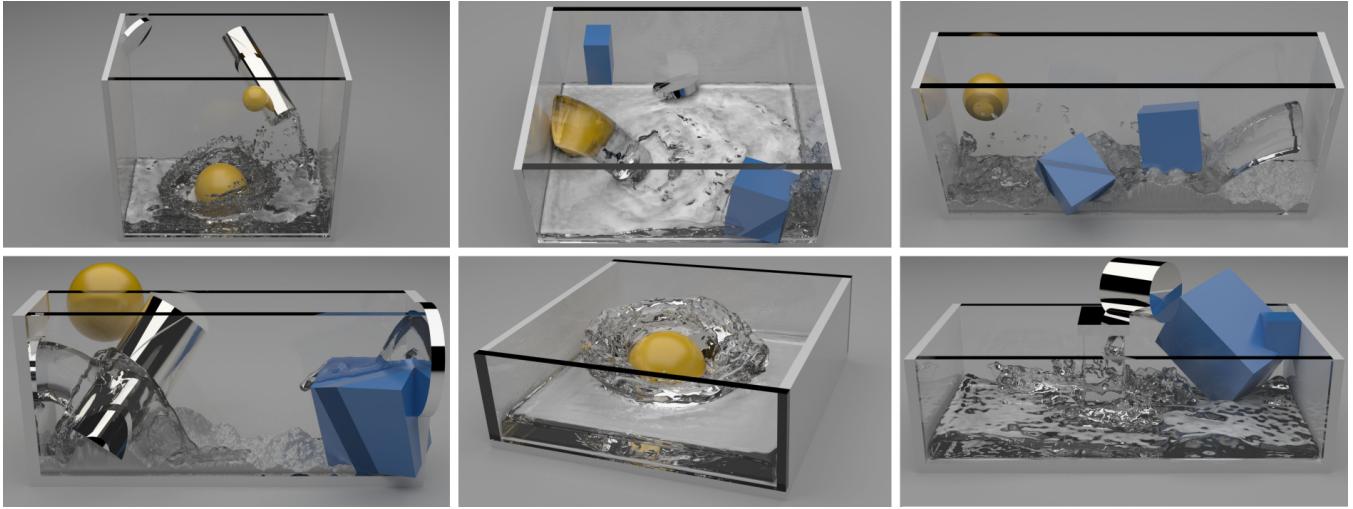
**Speeding up the method** Further analysis of the trees revealed, that viscosity and surface tension are rarely picked as the most discriminative features in the tree nodes. Using only pressure forces and compressibility the evaluation over particles is equivalent to the trilinearly interpolated evaluation over the grid cells with nonzero density. Using this approach it is possible to subsample the grid, and thus get a significant speed-up. At the  $2 \times 2 \times 2$  subsampling (used for example also in the FLIP method [Zhu and Bridson 2005]) we did not observe any additional approximation artifacts. Viscosity and surface tension can be then included as external forces over the grid. This approach led to qualitatively similar results at much

	1M	2M	4M	8M
Obstacle distance map	4.93	8.80	16.56	17.79
Integral volume calculation	5.78	9.71	21.57	69.95
Particles to grid interpolation	1.00	1.54	3.00	5.21
Regression forest evaluation	4.87	9.87	19.40	37.03
Grid to particles interpolation	3.41	6.48	11.44	20.98
Collision Detection	1.72	3.07	6.05	10.97
CPU-GPU data transfer	2.37	4.61	10.24	18.50
All	24.08	44.08	88.26	161.93

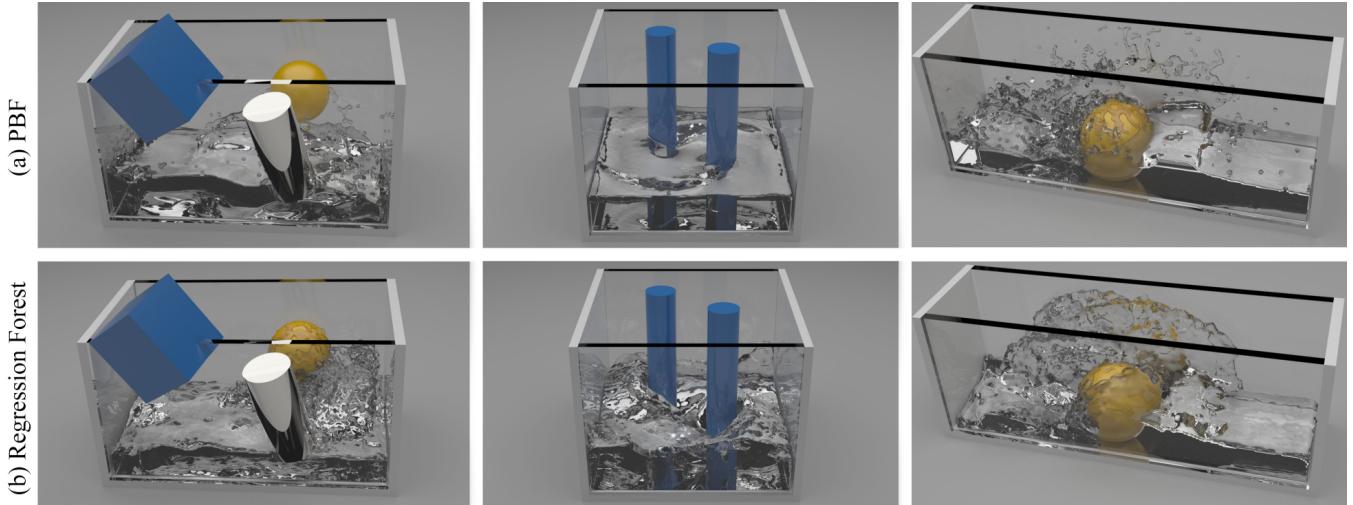
**Table 2:** Runtime per frame of different components of our real-time GPU pipeline on NVidia GTX 780 depending on the number of particles for a simulation with 3 moving obstacles (one box, one sphere and one cylinder) with  $2 \times 2$  grid subsampling. All numbers are for one particular simulation; the running time depends also on the number, type and size of obstacles (Obstacle distance map, collision detection), size of the space (Integral volume calculation), subsampling and the size of space occupied by particles (regression forest evaluation). The obstacle distance map does not need to be recalculated for fixed obstacles. The running time of each individual component grows approximately linearly with the size of the simulation. All results were obtained using one frame ( $\Delta t = 1/32$  s) as a simulation step.

lower computational cost and memory requirements, because it requires only one integral volume for the density to be calculated. Eventually, we trained 3 regression trees with this approach and used it for all our real-time GPU simulations. The snapshots of our test set simulations rendered using Mitsuba renderer [Jakob 2010] are shown in Figure 4. The snapshots of comparisons with the PBF solutions are shown in Figure 5. Our method provides results of reasonable quality at the running time order(s) of magnitude faster than any existing solution. The snapshots of our real-time simulations, each one with 1 to 1.5 million particles, with user controlled obstacles are shown in Figure 6. Our simulations showed the potential to be applied in interactive applications with user interactions. The running time comparisons to PCISPH and PBF is shown in Table 1. The running time of each component for a million particles for one particular scene with 3 obstacles is shown in Table 2. The times depend on several other parameters, such as the size of the space, type and size of obstacles etc.

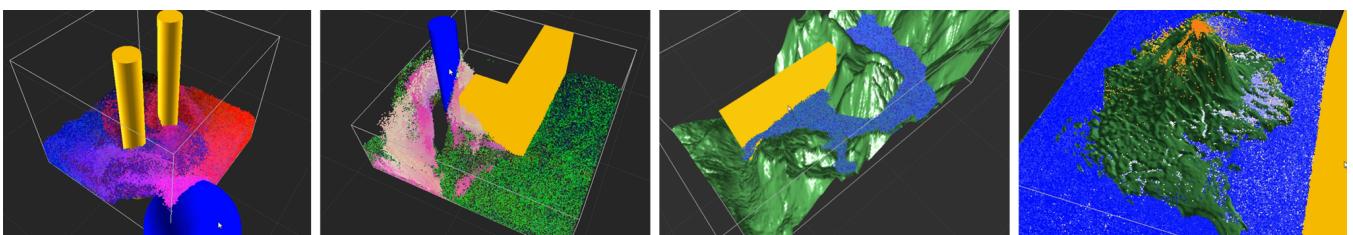
**Analysis of the weaknesses of the method** The main problem of our method is the same as the weakness of all machine learning approaches; the learning methods are not capable to extrapolate the model far outside the data observed during training. This applies to density, viscosity, particle radius, height of the simulation space (the distribution of boxes needs to be adapted) and time step (for correction approach only). The learning method is independent on external forces, however, the discriminativeness of features often depends on it. This can be observed for example in the distribution of boxes chosen during training. It does not have a zero mean, but it is shifted down due to statistically more probable interaction of fluid particles with a floor rather than with a roof. Over-fitting to observed data has also some good sides; the algorithm provides a very stable non-divergent solution even for cases where the incompressibility constraint can not be fulfilled due to moving obstacles. A conceptual weakness of our approach is the lack of rotation invariance, which is not directly enforced but only induced using lots of random rotations of the training data. Enforcement of such constraint would make the integral features inapplicable and thus significantly increase the running time. Another practical problem of our method is that water surfaces are not always perfectly flat due to imprecise predictions of our regressor.



**Figure 4:** The obtained results using our regression forest method (rendered offline). Our learning approach managed to successfully approximate a large state-space of a behaviour of fluid particles without any significant artifacts. Our method showed the potential to be useful for an application where running time matters more than the physical exactness, such as in computer games or interactive design.



**Figure 5:** Comparison with PBF. Our method produced slightly different, but globally similar visually plausible results. Simulations usually differ for any two different solvers or configurations of a particular solver.



**Figure 6:** Realtime interactive experiments with user interaction. The experiments showed that our method can be used in interactive realtime frameworks for fluid simulations with up to 2 million particles. At the moment there are no other alternatives that provide comparably fast visually pleasant stable solutions at such scales.

## 5 Conclusions

In this paper we proposed a novel machine learning approach to learn the fluid dynamics from a set of example simulations. We designed a set of translation and inertia invariant discriminative features, which correspond to individual forces and the incompressibility constraint of the Navier-Stokes equations, and trained a regressor, capable of iteratively predicting the next state of each particle. Our approach showed the potential to be a good replacement of standard solvers in settings, where running times is more important than the exactness of a simulation, such as in computer games or interactive design. We believe our promising results will encourage other researchers to investigate and possibly improve the performance. We also explored several other lines of thoughts, which despite being well motivated, turned out to be inferior to the final proposed solution. We believe, that these negative experiments also contribute to the more complete picture about the application of machine learning to physics-based simulations.

In our future work we want to try to apply our learning approach to other similar physics-based problems, such as smoke, fire, cloth sand, snow, elastic body simulations, etc. Another interesting direction is to combine learning methods with standard solvers to obtain both fast and highly accurate simulations.

## References

- BREIMAN, L. 2001. Random forests. In *Machine Learning*.
- CHENTANEZ, N., AND MÜLLER, M. 2010. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 197–206.
- CHENTANEZ, N., AND MÜLLER, M. 2011. Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Transactions on Graphics (Proceedings SIGGRAPH)* 30, 82:1–82:10.
- DE WITT, T., LESSIG, C., AND FIUME, E. 2012. Fluid simulation using laplacian eigenfunctions. *ACM Trans. Graph.* 31, 1, 10:1–10:11.
- DESBRUN, M., AND CANI, M.-P. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, Springer-Verlag, 61–76.
- ENRIGHT, D., FEDIKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics* 183, 1, 83–116.
- GUPTA, M., AND NARASIMHAN, S. G. 2007. Legendre fluids: A unified framework for analytic reduced space modeling and rendering of participating media. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation* (2007), D. Metaxas and J. Popovic, Eds.
- HORVATH, C. J., AND SOLENTHALER, B., 2013. Mass preserving multi-scale SPH. Pixar Technical Memo 13-04, Pixar Animation Studios.
- IHMSEN, M., CORNELIS, J., SOLENTHALER, B., HORVATH, C., AND TESCHNER, M. 2013. Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics*. doi:10.1109/TVCG.2013.105.
- JAKOB, W., 2010. Mitsuba renderer. <http://www.mitsuba-renderer.org>.
- KIM, T., AND DELANEY, J. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics (Proceedings SIGGRAPH)* 32, 4, 62:1–62:9.
- LADICKY, L., RUSSELL, C., KOHLI, P., AND TORR, P. H. S. 2009. Associative hierarchical CRFs for object class image segmentation. In *International Conference on Computer Vision*.
- LUCY, L. 1977. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal* 82, 1013–1024.
- MACKLIN, M., AND MUELLER, M. 2013. Position based fluids. *ACM Transactions on Graphics (Proceedings SIGGRAPH)* 32, 1–5.
- MONAGHAN, J. 1992. Smoothed particle hydrodynamics. *Ann. Rev. Astron. Astrophys.* 30, 543–574.
- MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 154–159.
- RAVEENDRAN, K., WOJTAN, C., THUERÉY, N., AND TURK, G. 2014. Blending liquids. *ACM Trans. Graph.* 33, 4 (July), 137:1–137:10.
- SHOTTON, J., WINN, J., ROTHER, C., AND CRIMINISI, A. 2006. *TextronBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European Conference on Computer Vision*.
- SHOTTON, J., FITZGIBBON, A., COOK, M., AND BLAKE, A. 2011. Real-time human pose recognition in parts from single depth images. In *Conference on Computer Vision and Pattern Recognition*.
- SHOTTON, J., GLOCKER, B., ZACH, C., IZADI, S., CRIMINISI, A., AND FITZGIBBON, A. 2013. Scene coordinate regression forests for camera relocalization in rgbd images. In *Conference on Computer Vision and Pattern Recognition*.
- SOLENTHALER, B., AND PAJAROLA, R. 2009. Predictive-corrective incompressible SPH. *ACM Transactions on Graphics (Proceedings SIGGRAPH)* 28, 40:1–40:6.
- STANTON, M., HUMBERTON, B., KASE, B., O'BRIEN, J. F., FATAHALIAN, K., AND TREUILLE, A. 2014. Self-refining games using player analytics. *ACM Transactions on Graphics (Proceedings SIGGRAPH)* 33, 4, 73:1–73:9.
- TAYLOR, J., SHOTTON, J., SHARP, T., AND FITZGIBBON, A. 2012. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Conference on Computer Vision and Pattern Recognition*.
- THÜREY, N., WOJTAN, C., GROSS, M., AND TURK, G. 2010. A multiscale approach to mesh-based surface tension flows. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 29, 3.
- TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. In *ACM Transactions on Graphics (Proceedings SIGGRAPH)*.
- VIOLA, P., AND JONES, M. 2004. Robust real-time face detection. *International Journal of Computer Vision*.
- WICKE, M., STANTON, M., AND TREUILLE, A. 2009. Modular bases for fluid dynamics. *Transactions on Graphics* 28, 3.
- ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (Proceedings SIGGRAPH)* 24, 965–972.