Dynamic Upsampling of Smoke through Dictionary-based Learning

KAI BAI, ShanghaiTech University/SIMIT/UCAS
WEI LI, ShanghaiTech University/SIMIT/UCAS
MATHIEU DESBRUN, Caltech/ShanghaiTech University
XIAOPEI LIU, ShanghaiTech University

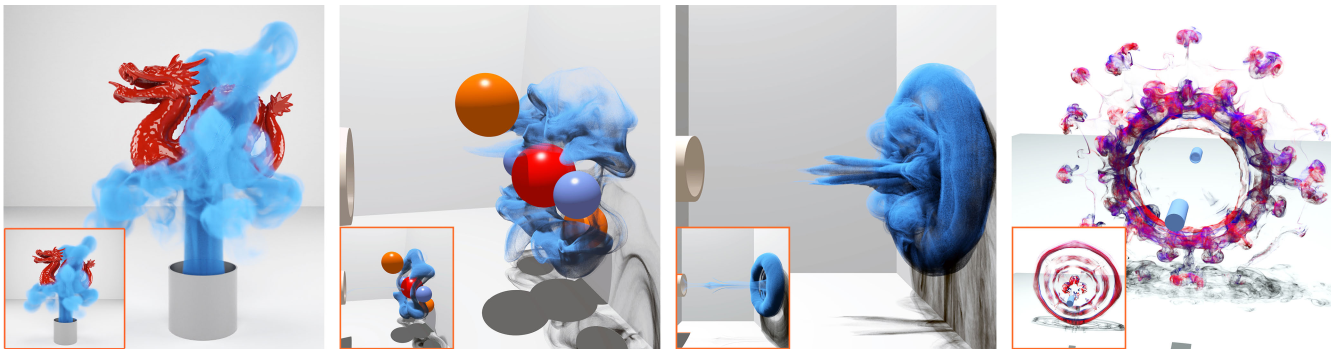


Fig. 1. **Fine smoke animations from low-resolution simulation inputs.** We present a versatile method for dynamic upsampling of smoke flows from low-resolution inputs (shown here as insets) which handles a variety of animation contexts: (a) *super-resolution*, where a spatially-downsampled animation (here, a smoke puff on a dragon obstacle) is upsampled, even if our neural network was only trained on a flow over a simple ball; (b) *generalized upsampling*, where a few pairs of low-res and high-res simulations were used as a training set (here, a smoke simulation with a single ball obstacle, and another one with 5 balls arranged in a vertical plane), from which one can synthesize a high-res simulation from any coarse input (here, 5 balls at a 45° angle compared to the second training simulation); (c) *restricted upsampling*, where the training set contains only a few sequences (here, 4 simulations with various inlet sizes), leading to faster training and more predictive synthesis results (the input uses an inlet size not present in the training set); and (d) *re-simulation*, where the training set consists of only one simulation, from which we can quickly "re-simulate" the original flow very realistically for small variations of the original animation (here, vortex rings colliding). Depending on the chosen context, our synthesized smoke animations either efficiently add visually plausible details, or reconstruct physically-based fine structures close to the associated fine simulation — but an order of magnitude faster.

Simulating turbulent smoke flows with fine details is computationally intensive. For iterative editing or simply faster generation, efficiently upsampling a low-resolution numerical simulation is an attractive alternative. We propose a novel learning approach to the dynamic upsampling of smoke flows based on a training set of flows at coarse and fine resolutions. Our multiscale neural network turns an input coarse animation into a sparse linear combination of small velocity patches present in a precomputed over-complete dictionary. These sparse coefficients are then used to generate a high-resolution smoke animation sequence by blending the fine counterparts of the coarse patches. Our network is initially trained from a sequence of example simulations to both construct the dictionary of corresponding coarse and fine patches and allow for the fast evaluation of a sparse patch encoding of any coarse input. The resulting network provides an accurate upsampling when the coarse input simulation is well approximated by patches present in the training set (e.g., for re-simulation), or simply visually-plausible upsampling when input and training set differ significantly. We show a variety of examples to ascertain the strengths and limitations of our approach, and offer comparisons to existing approaches to demonstrate its quality and effectiveness.

CCS Concepts: • **Computing Methodologies** → **Neural Networks; Physical Simulation**.

Additional Key Words and Phrases: Fluid Simulation, Dictionary Learning, Neural Networks, Smoke Animation

ACM Reference Format:
Kai Bai, Wei Li, Mathieu Desbrun, and Xiaopei Liu. 2020. Dynamic Upsampling of Smoke through Dictionary-based Learning. *ACM Trans. Graph.* 1, 1, Article 1 (January 2020), 19 pages. https://doi.org/10.1145/3412360

1 INTRODUCTION

Visual simulation of smoke is notoriously difficult due to its highly turbulent nature, resulting in vortices spanning a vast range of space and time scales. As a consequence, simulating the dynamic behavior of smoke realistically requires not only sophisticated non-dissipative numerical solvers [Li et al. 2020; Mullen et al. 2009; Qu et al. 2019; Zhang et al. 2015], but also a spatial discretization with sufficiently high resolution to capture fine-scale structures, either uniformly [Kim et al. 2008b; Zehnder et al. 2018] or adaptively [Losasso et al. 2004; Weißmann and Pinkall 2010a; Zhang et al.

Authors' addresses: Kai Bai, Wei Li, Xiaopei Liu, School of Information Science and Technology (Shanghai Engineering Research Center of Intelligent Vision and Imaging) of ShanghaiTech University, Shanghai, China; Kai Bai and Wei Li are also affiliated with the Shanghai Institute of Microsystem and Information Technology (SIMIT) and the University of the Chinese Academy of Sciences (UCAS); Mathieu Desbrun, California Institute of Technology, Pasadena (CA), USA, on sabbatical at SIST in ShanghaiTech University, Shanghai, China.

0730-0301/2020/1-ART1 $15.00
https://doi.org/10.1145/3412360

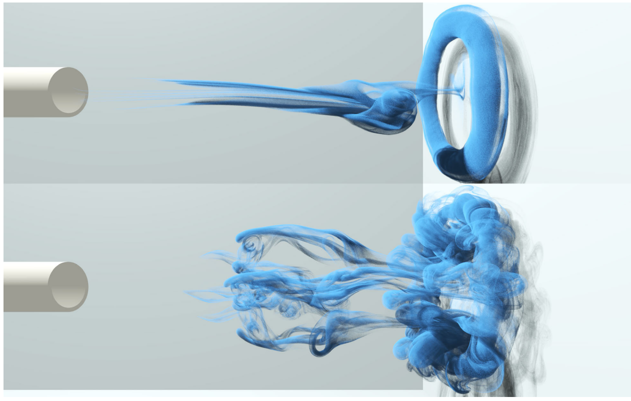


Fig. 2. **Coarse vs. fine smoke simulations.** A smoke simulation computed using a low (top: $50 \times 75 \times 50$) vs. a high resolution (bottom: $200 \times 300 \times 200$) respectively, for the same Reynolds number (5000). Flow structures are visually quite distinct since different resolutions resolve different physical scales, thus producing quite different instabilities.

2016]. This inevitably makes such direct numerical simulations computationally intensive.

In order to compromise between efficiency and visual realism for large-scale scenes, the general concept of physics-inspired upsampling of dynamics [Kavan et al. 2011] can be leveraged: low-resolution simulations can be computed first, from which a highly-detailed flow is synthesized using fast procedural models that are only loosely related with the underlying fluid dynamics, e.g., noise-based [Bridson et al. 2007; Kim et al. 2008a] or simplified turbulence models [Pfaff et al. 2010; Schechter and Bridson 2008]. Very recently, machine learning has even been proposed as a means to upsample a coarse flow simulation [Chu and Thuerey 2017] (or even a down-sampled flow simulation [Werhahn et al. 2019; Xie et al. 2018]) to obtain finer and more visually-pleasing results inferred from a training set of actual simulations. However, while current upsampling methods can certainly add visual complexity to a coarse input, the synthesized high-resolution fluid flow often fails to exhibit the type of structures that the original physical equations are expected to give rise to: the inability to inject physically-consistent small-scale vortical structures leads to visual artifacts, making the resulting flow simulations less realistic.

Synthesizing a high-resolution flow from a low-resolution input is fundamentally difficult because both their local and global structures may differ significantly (see Fig. 2 for an example). Discrepancy between low-res and high-res numerical simulations is not only due to discretization errors, but also to flow instabilities, becoming more pronounced for high Reynolds number turbulent flows. However, one could hope that many of the details of the flow can be inferred by comparing local patches of the input coarse flow to a catalog of existing low-res/high-res pairs of numerical simulations: a proper local encoding of existing upsampled sequences could help predict the appearance and evolution of fine structures from existing coarse input simulations — in essence, learning how the natural shedding of small vortices appears from the corresponding coarse simulation.

In this paper, we propose to upsample smoke motions through *dictionary learning* [Garcia-Cardona and Wohlberg 2018] (a common approach in image upsampling [Yang et al. 2010]) based on the observation that although turbulent flows look complex, local structures and their evolution do not differ significantly as they adhere to the self-advection process prescribed by the fluid equations: local learning through sparse coding followed by synthesis through a dictionary-based neural network is thus more appropriate than global learning methods such as convolutional neural networks [Tompson et al. 2017].

1.1 Related Work

Smoke animation has been widely studied for more than two decades in computer graphics. We review previous work relevant to our contributions, covering both traditional simulation of smoke and data-driven approaches to smoke animation.

Numerical smoke simulation. Smoke animation has relied most frequently on numerical simulation of fluids in the past. Fast fluid solvers [Stam 1999], and their higher-order [Kim et al. 2005; Selle et al. 2008], momentum-preserving [Lentine et al. 2010] or advection-reflection [Zehnder et al. 2018] variants, can efficiently simulate smoke flows on uniform grids. However, creating complex smoke animations requires relatively high resolutions to capture fine details. Unstructured grids [Ando et al. 2013; de Goes et al. 2015; Klingner et al. 2006; Mullen et al. 2009] and adaptive methods, where higher resolutions are used in regions of interest and/or with more fluctuations [Losasso et al. 2004; Setaluri et al. 2014; Zhu et al. 2013] have been proposed to offer increased efficiency — but the presence of smoke turbulence in the entire domain often prevents computational savings in practice. On the other hand, particle methods, e.g, smoothed particle hydrodynamics [Akinci et al. 2012; Becker and Teschner 2007; Desbrun and Gascuel 1996; Ihmsen et al. 2014; Peer et al. 2015; Solenthaler and Pajarola 2009; Winchenbach et al. 2017] and power particles [de Goes et al. 2015] can easily handle adaptive simulations. However, a large number of particles are necessary to obtain realistic smoke animations to avoid poor numerical accuracy for turbulent flows. Hybrid methods [Jiang et al. 2015; Raveendran et al. 2011; Zhang and Bridson 2014; Zhang et al. 2016; Zhu and Bridson 2005], which combine both particles and grids, can be substantially faster, but particle-grid interpolations usually produce strong dissipation unless polynomial basis functions are used for improved numerics [Fu et al. 2017]. These methods remain very costly in the context of turbulent smoke simulation. Another set of approaches that can simulate smoke flow details efficiently are vortex methods [Golas et al. 2012; Park and Kim 2005]; in particular, vortex filaments [Weißmann and Pinkall 2010b] and vortex sheets [Brochu et al. 2012; Pfaff et al. 2012] are both effective ways to simulate turbulent flows for small numbers of degrees of freedom, and good scalability can be achieved with fast summation techniques [Zhang and Bridson 2014]. However, no existing approach has been proposed to upsample low-resolution vortex-based simulations to full-blown high-resolution flows to further accelerate fluid motion generation. We note finally that a series of other numerical methods have been developed to offer efficiency through the use of other fluid models [Chern et al. 2016] or of massively-parallelizable mesoscopic models like the lattice Boltzmann method [Chen and Doolen 1998; De Rosis 2017; d'Humières 2002; Geier et al. 2006; Li et al. 2019, 2020; Liu et al. 2014; Lycett-Brown et al. 2014], but here again, the ability to run only a coarse simulation to quickly generate a high-resolution fluid motion has not been investigated.

Early upsampling attempts. Over the years, various authors have explored ways to remediate the shortcomings induced by numerical simulation on overly coarse grids in the hope of recovering high-resolution results. Reinjecting fine details through vorticity confinement [Fedkiw et al. 2001; John Steinhoff 1994], Kolmogorov-driven noise [Bridson et al. 2007; Kim et al. 2008b], vorticity correction [Zhang et al. 2015], smoke style transfer [Sato et al. 2018] or modified turbulence models [Pfaff et al. 2010; Schechter and Bridson 2008] partially helps, but visually important vortical structures are often lost: none of these approaches provides a reliable way to increase the resolution substantially without clearly deviating from the corresponding simulation on a fine computational grid.

Data-driven approaches. Given the computational complexity of smoke animation through numerical simulation, data-driven approaches have started to emerge in recent years as a promising alternative. Some techniques proposed generating a flow field entirely based on a trained network, completely avoiding numerical simulation for fluid flows. [Guo et al. 2016; Kim et al. 2019] and [Jeong et al. 2015] proposed a regression forest based approach for learning SPH simulation. [Tompson et al. 2017] trained a neural network to predict pressure without solving a Poisson equation, while [Umetani and Bickel 2018] proposed to predict aerodynamic forces and velocity/pressure fields from an inflow direction and a 3D shape. A few data-driven approaches directly synthesized flow details for smoke and liquid animations from low-resolution simulations instead: e.g., [Chu and Thuerey 2017; Werhahn et al. 2019; Xie et al. 2018] created high frequency smoke details based on neural networks, while [Um et al. 2018] modeled fine-detail splashes for liquid simulations from existing data. Yet, these recent data-driven upsampling approaches do not generate turbulent smoke flows that are faithful to their physical simulations using similar boundary conditions: the upsampling of a coarse motion often fails to reconstruct visually-expected details such as leapfrogging in vortex ring dynamics, even if the coarse motion input is quite similar to exemplars from the training set. Our work focuses on addressing this deficiency via a novel neural network based on dictionary learning.

1.2 Overview

Although smoke flows exhibit intricate structures as a whole, the short-term evolution of a *local patch* of smoke only follows a restricted gamut of behaviors, and the complexity of high resolution turbulent flow fields emerges from the rich combination of these local motions. This patch-based view of the motion motivates the idea of *dictionary learning*, as used in image upsampling, to achieve physically-driven upsampling for coarse smoke flows.

However, existing dictionary learning methods for image upsampling cannot be directly used for flow synthesis. First and foremost, we have to learn structures from vector fields (or vortex fields) instead of scalar fields, a much richer set than typical image upsampling methods are designed for. Second, we are dealing with a dynamical system instead of a static image, so we must also adapt the actual upsampling process appropriately.

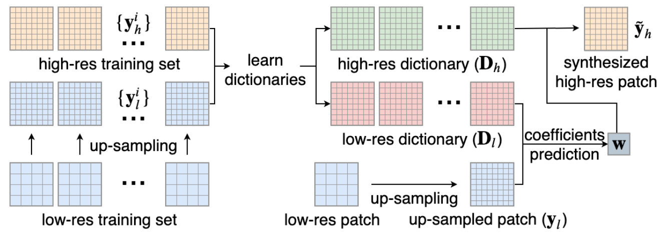


Fig. 3. **Dictionary learning for image upsampling.** In order to synthesize high-resolution images, one prepares a training set of local patch pairs ($\mathbf{y}_l^i$ and $\mathbf{y}_h^i$) from low and high resolution images respectively (left), from which we can learn two dictionaries ($\mathbf{D}_l$ and $\mathbf{D}_h$). Given a low resolution image, each coarse patch is then used to predict a set of sparse coefficients $\mathbf{w}$ such that the corresponding patch in the high-resolution image can be synthesized using $\mathbf{D}_h$ and the same sparse coefficients $\mathbf{w}$.

In this paper, we propose a novel neural network structure for dictionary learning of smoke flow upsampling, where the motion of fine fluid flow patches is learned from their coarse versions. We ensure good spatial and temporal coherence by directly learning from the high-resolution residuals between coarse motion predictions and actual fine motion counterparts. Plausible high-resolution flows can then be quickly synthesized from low-resolution simulations, providing much richer dynamics and higher efficiency (often an order of magnitude faster) than existing data-driven methods. We demonstrate that our approach produces visually-complex upsampling of coarse smoke flow simulation through local and physically-driven interpolation between (and to a certain extent, extrapolation from) the training examples. In particular, we show that our results offer a better approximation to the real fine-scale dynamics if the coarse input does not deviate too much from the training set as in the case of re-simulation; if the coarse input is far from the exemplars in the training set, our upsampling technique produces a visually-plausible (but not physically-accurate) high-resolution flow capturing the fine motion better than the state-of-the art methods. Fig. 1 shows examples of animation results generated from our upsampling with four different types of training sets based on our dictionary learning approach, where coarse simulations are upsampled by a factor of 64 (4×4×4), exhibiting vortex structures that a (significantly more) costly fine simulation would typically exhibit. We also evaluate our results in terms of visual quality, energy spectrum, synthesis error compared to fine simulations, as well as computing performance in order to thoroughly validate the advantages of our method.

2 BACKGROUND ON DICTIONARY LEARNING

We first review traditional dictionary learning for image up-sampling, as its foundations and algorithms will be key to our contributions once properly adapted to our animation context.

2.1 Foundations

In image upsampling, a high-resolution image is synthesized from a low-resolution image with a learned dictionary of local patches as summarized in Fig. 3: the input low-resolution image is first written as a sparse weighted sum of local "coarse" patches; then the resulting high-resolution image is written as the weighted sum, with exactly the same weights, of the corresponding "fine" (upsampled) patches, when each corresponding pair of coarse and fine patches comes from a training set of upsampled examples. One thus needs to find a dictionary of patch pairs, and a way to write any low-resolution image as a linear combination of coarse dictionary patches.

Role of a dictionary. A dictionary for image upsampling is a set of upsampled low-resolution local patches $\{\mathbf{d}_i^l\}_i$ of all the same size, and their associated high-resolution local patches $\{\mathbf{d}_i^h\}_i$ (for instance, all of the size 5×5 pixels). By storing the dictionary patches as vectors, any upsampled coarse patch $\mathbf{y}_l$ can be approximated by a patch $\tilde{\mathbf{y}}_l$ that is a sparse linear combination of coarse dictionary patches, i.e., $\tilde{\mathbf{y}}_l = \sum_i w_i \mathbf{d}_l^i$ with a sparse set of coefficients w_i. An upsampled patch $\tilde{\mathbf{y}}_h$ corresponding to the input upsampled coarse patch $\mathbf{y}_l$ can then be expressed as $\tilde{\mathbf{y}}_h = \sum_i w_i \mathbf{d}_h^i$. For convenience, we will denote by $\mathbf{D}_l = (\mathbf{d}_l^1 ... \mathbf{d}_l^N)$ the matrix storing all the coarse dictionary patches (where each patch is stored as a vector), and similarly for all the high-resolution dictionary patches using $\mathbf{D}_h$ — such that a patch $\tilde{\mathbf{y}}_l$ (resp., $\tilde{\mathbf{y}}_h$) can be concisely computed as $\mathbf{D}_l\mathbf{w}$ (resp., $\mathbf{D}_h\mathbf{w}$) where $\mathbf{w} = (w_1, ..., w_N)$.

Finding a dictionary. For a given training set of coarse and fine image pairs, we can find a dictionary with N patch elements by making sure that it not only captures all coarse patches well, but its high-resolution synthesis also best matches the input fine images. If we denote by $\mathbf{Y}_l$ the vector storing all the coarse patches $\mathbf{y}_l$ available from the training set and by $\mathbf{Y}_h$ the vector of their corresponding fine patches $\mathbf{y}_h$, the dictionaries as well as the sparse weights are found through a minimization [Yang et al. 2010]:

$$\underset{\mathbf{D}_l,\mathbf{D}_h,\mathbf{w}}{\operatorname{argmin}} \ \frac{1}{2}\left\|\mathbf{Y}_l - \widetilde{\mathbf{Y}}_l(\mathbf{D}_l)\right\|_2^2 + \left\|\mathbf{Y}_h - \widetilde{\mathbf{Y}}_h(\mathbf{D}_h)\right\|_2^2 + \lambda\left\|\mathbf{w}\right\|_1 , \quad (1)$$

which evaluates the representative power of the coarse dictionary patches (through the first term) and the resulting upsampling quality (using the ℓ_2 difference between the upsampled patches $\widetilde{\mathbf{Y}}_h$ and their ground-truth counterparts $\mathbf{Y}_h$) while penalizing non-sparse weights via an ℓ_1 norm of $\mathbf{w}$ times a regularization factor λ. Solving for the dictionary patches minimizing this functional is achieved using the K-SVD method [Aharon et al. 2006].

Upsampling process. Once the optimization stage has returned a dictionary, upsampling a low-resolution input image is done by finding a sparse linear combination of the coarse dictionary patches for each local patch of the input. The method of orthogonal matching pursuit (OMP) is typically used to find the appropriate (sparse) weights that best reproduce a local patch based on the dictionaries (other pursuit methods used in graphics can potentially be used too [Teng et al. 2015; Von Tycowicz et al. 2013]), from which the high-resolution patch is directly reconstructed using these weights, now applied to the high-resolution dictionary patches. The final high-resolution image is generated by blending all locally synthesized high-resolution patches together.

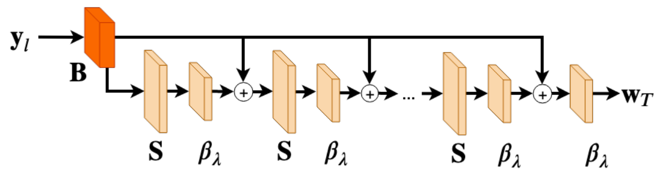


Fig. 4. **Original LISTA network.** Illustration of feed-forward LISTA neural network [Gregor and LeCun 2010] with T layers.

2.2 Scalable solver to find sparse linear combinations

Looking ahead at our task at hand, the fact that we will have to deal with 3D vector-based fields will make the dimension of the vectors $\mathbf{y}_l$ quite larger than what is typically used in image upsampling. In this case, the OMP-based optimization required to perform upsampling will become extremely slow, and may even return poor results as shown in Fig. 5(b). Thus, a more efficient method to compute $\mathbf{w}$ given a low-resolution patch is required. One scalable approach is the LISTA neural network proposed by Gregor and LeCun [2010]: it is a learning-based formulation derived from the iterative shrinkage thresholding algorithm (ISTA) [Daubechies et al. 2004] using the following iteration process:

$$\mathbf{w}_{t+1} = \beta\big(\mathbf{S}\mathbf{w}_t + \mathbf{B}\mathbf{Y}\,;\lambda\big) , \quad (2)$$

where $\mathbf{B} = h\mathbf{D}^T$, $\mathbf{S} = \mathbb{I} - \mathbf{BD}$ ($\mathbb{I}$ being the identity matrix) with $\mathbf{D} = [\mathbf{D}_l; \mathbf{D}_h]^T$ the matrix concatenating the coarse and fine dictionary patches, $\mathbf{Y} = [\mathbf{Y}_l; \mathbf{Y}_h]^T$, h is the iteration step size, and $\beta(\cdot\,;\cdot)$ is a vector function constructed to favor the sparsity of $\mathbf{w}$:

$$\beta_i(\mathbf{x}\,;\lambda) = \operatorname{sgn}(x_i)\ \max\{|x_i| - \lambda, 0\} , \quad (3)$$

where β_i denotes the i-th component of the output vector from the vector function β and x_i represents the i-th component of vector $\mathbf{x}$. This immediately corresponds to a feed-forward neural network with T layers, see Fig. 4, where t in Eq. (2) denotes the t-th layer in the network, and β is the activation function of the network.

In the original LISTA algorithm, in order to enable better prediction of the final $\mathbf{w}$, $\mathbf{B}$ and $\mathbf{S}$ are both updated during learning while λ remains fixed. The number of layers in the network corresponds to the total number of iterations in Eq. (2), with more layers typically leading to more accurate prediction. The network is trained using a set of pairs $\{(\mathbf{y}_i, \mathbf{w}_i), i = 1, 2, ..., k.\}$, whose weights are computed using the OMP method on a given learned dictionary $\mathbf{D}$, and with a loss function for the network defined as:

$$L_T(\Theta) = \frac{1}{k}\sum_{i=1}^{k}\|\mathbf{w}_T(\mathbf{y}_i;\Theta) - \mathbf{w}_i\|_2^2 , \quad (4)$$

where T is the index of the output layer, and Θ is the vector assembled from all parameters (including $\mathbf{B}$, $\mathbf{S}$ and λ from Eq. (2)). After training, we can use the learned Θ to predict $\mathbf{w}$ from an input $\mathbf{y}_l$ by going through the whole network. This LISTA-based solve provides much higher efficiency than the traditional OMP approach.

2.3 Inadequacy of direct upsampling on coarse simulation

While the idea of upsampling a coarse smoke motion in a frame-by-frame fashion through a direct application of the image upsampling approach presented above sounds attractive, it is bound to fail for a number of reasons. First, the issue of coherency over time is not addressed at all: the weights used for a given spatial region over two successive frames have no reasons to bear any resemblance to each

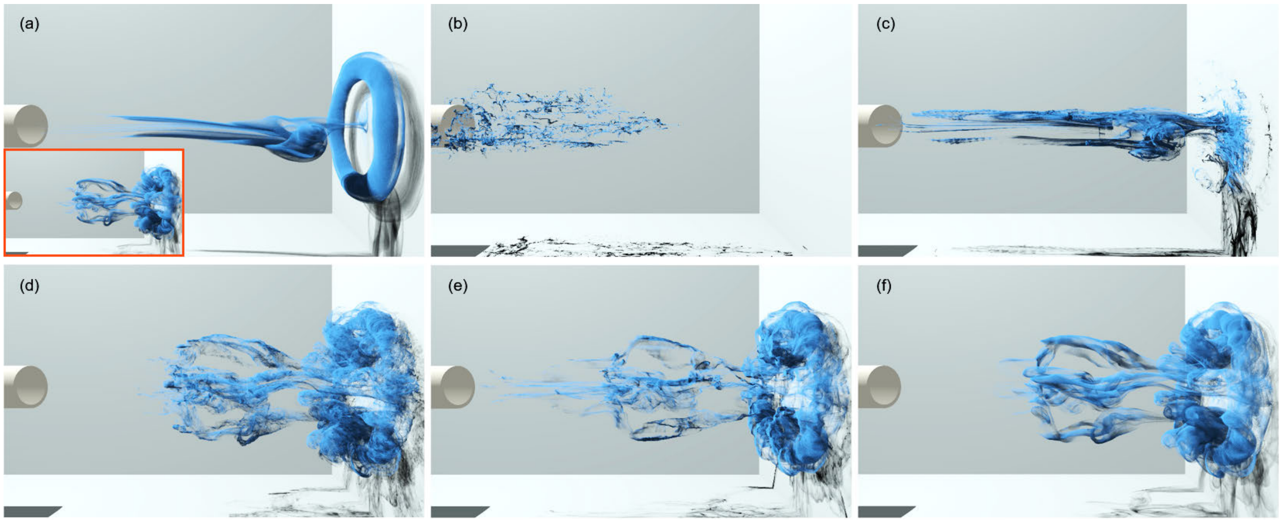


Fig. 5. **Strategies for upsampling.** From an input low-resolution simulation in (a) (its fine simulation counterpart is shown as inset), the most straightforward method to upsample it is to use the traditional dictionary learning from image up-sampling, this time in 3D, using either the OMP method (b) or a LISTA network (c) — but because of how different low and high resolution simulations look, they both produce poor synthesis results; reformulating the problem as a dictionary learning reconstruction, we can obtain much better result in (d), although it tends to be slightly noisy; with our novel network and by representing each local patch with velocities only (e), the results exhibit spatial and temporal incoherence; when adding to each local patch spatial and time codes (f), a coherent synthesis can be obtained, looking quite close to the fine simulation counterpart.

other, thus potentially creating motion flickering. Second, one major difference between the image upsampling problem and our smoke upsampling target is that the coarse and fine simulations can differ widely due to the chaotic changes that turbulence naturally exhibits (Fig. 2). Indeed, image upsampling relies on an energy (Eq. (1)) which puts coarse approximation and fine approximation errors on equal footing, and the LISTA approach based on the iterative optimization of Eq. (2) may also fail to converge to the right OMP solution if $\mathbf{y}_l$ and $\mathbf{y}_h$ differ significantly in structure, since it performs essentially a local search (see Fig. 5(c) for such an experiment). This suggests a change of both the objective for smoke upsampling and the LISTA network formulation. We present our approach next based on the idea that the coarse simulation can be used as *a predictor of the local motion*, from which *the correction needed to get a high resolution frame* is found through dictionary learning.

3 SMOKE UPSAMPLING VIA DICTIONARY LEARNING

We now delve into our approach for smoke upsampling. We provide the intuition behind our general approach, before detailing the new neural network. Finally, we discuss how we augment the patch representation to offer a spatially and temporally coherent upsampling of smoke flows, and provide a simple method to have a better training of our upsampling network.

3.1 Our approach at a glance

Based on the relevance of dictionary learning to the upsampling of smoke flows, but given the inadequacy of the current techniques to the task at hand, we propose a series of changes in the formulation of the upsampling problem.

Prediction/correction. In our dynamical context, it is often beneficial to consider a motion as a succession of changes in time. We thus formulate the upsampling of a coarse patch $\mathbf{y}_l$ as a patch $\tilde{\mathbf{y}}_h = \text{up}(\mathbf{y}_l) + \Delta_h$, where $\text{up}(\cdot)$ corresponds to a straightforward spatial upsampling through direct enlargement of the coarse patch, and Δ_h is a residual high-resolution patch. This amounts to a predictor-corrector upsampling, where the coarse patch is first upsampled straightforwardly by $\text{up}(\cdot)$ before details are added. The residual patches should not be expected to only have small magnitudes, though: as we discussed earlier, differences between coarse and fine simulations can be large in turbulent regions of the flow. Since we will use a dictionary of these high-frequency details, their magnitude has no negative influence, only their diversity matters.

Residual dictionary. Since a smoke flow locally follows the Navier-Stokes equations, we can expect that the residuals can be well expressed by a fine dictionary $\mathbf{D}_h$. This is indeed confirmed numerically: if one uses K-SVD to solve for a high-resolution dictionary (of 400 patches) with around 3M training patches from a single fine simulation, the dictionary-based reconstruction is almost visually perfect (albeit a little noisier) as demonstrated in Fig. 5(d), confirming that the local diversity of motion is, in fact, limited. We thus expect a residual Δ_h in our approach to be well approximated by a sparse linear combination of elements of a (high-resolution) dictionary $\mathbf{D}_h$, i.e., a residual is nearly of the form $\Delta_h \approx \mathbf{D}_h \mathbf{w}$. Just like in the case of image upsampling, sparsity of the weights is preferable as it avoids the unnecessary blurring introduced by the linear combination of too many patches.

Variational formulation. For efficiency reasons, we discussed in Sec. 2.3 that using a LISTA-based evaluation of the sparse weights is highly preferable to the use of OMP. This means that we need to train a network to learn to compute, based on a coarse input $\mathbf{y}_l$, the sparse weights $\mathbf{w}(\mathbf{y}_l)$. Thus, in essence, we wish to modify the traditional upsampling minimization of Eq. (1) to instead minimize

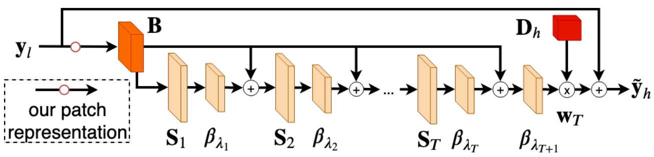


Fig. 6. **Our dictionary-based neural network.** We modify the original LISTA network of Fig. 4 by adding layer-specific matrices $\mathbf{S}_t$, regularization parameters λ_t as well as the residual dictionary as parameters to learn.

the errors in reconstruction of the type $\|\mathbf{y}_h - \text{up}(\mathbf{y}_l) - \mathbf{D}_h\mathbf{w}(\mathbf{y}_l)\|_2^2$ on a large series of training patches (with control over the sparsity of $\mathbf{w}$) *while also training a LISTA-like network for the weights*. Other notions of reconstruction errors, based on the vorticity, the difference of gradients, or even the divergence of the upsampled patches would also be good to incorporate in order to offer more user control over the upsampling process.

Based on these assumptions, we introduce a new neural network design, where learning a (high-resolution residual) dictionary and training a network to efficiently compute sparse linear coefficients are done simultaneously, thus requiring a single optimization.

3.2 Neural network design

Our proposed neural network follows mostly the structure of the LISTA network for sparse coding [Gregor and LeCun 2010], in the sense that it is also composed of several layers representing an iterative approximation of the sparse weights. Two key differences are introduced: first, we add more flexibility to the network by letting each of the T layers not only have its own regularization parameter λ_t, but also its own matrix $\mathbf{S}_t$; second, while the original LISTA network refers to the sparse weights $\mathbf{w}$ computed by the OMP method to define the loss, our loss will be measured based on the quality of the reconstruction engendered by the final weights and the dictionary $\mathbf{D}_h$ (the loss will be explicitly provided next in Sec. 3.3). Our fast approximation of sparse coding is achieved through the following modified LISTA-like iteration

$$\mathbf{w}_{t+1} = \beta\big(\mathbf{S}_t\mathbf{w}_t + \mathbf{B}\mathbf{y}\,;\lambda_t\big)\ , \tag{5}$$

where β is the same activation function as in Eq. (3) to enforce sparsity of $\mathbf{w}$. Our novel neural network, summarized in Fig. 6, can optimize all its network parameters (i.e., the residual dictionary $\mathbf{D}_h$, mapping matrices $\mathbf{S}_t$, regularization parameters λ_t, and the matrix $\mathbf{B}$) by the standard back-propagation procedure through the T different layers during learning as we will go over later.

3.3 Loss function design

To successfully guide our network during training, an important factor is the choice of loss function. Unlike the LISTA network for which the loss function from Eq. (4) requires a set of sparse coefficients $\mathbf{w}$, we construct our loss function directly based on the quality of synthesis results of our network.

ℓ_2 synthesis error. One measure for our loss function is the difference between an upsampled patch $\tilde{\mathbf{y}}_h^i$ found from a low-resolution patch $\mathbf{y}_l^i$ and the ground-truth high-resolution patch $\mathbf{y}_h^i$ from a training set containing K patches:

$$\mathcal{E}_\ell = \sum_{i=1}^{K} \left\|\mathbf{y}_h^i - (\mathbf{y}_l^i + \mathbf{D}_h\mathbf{w}_T(\mathbf{y}_l^i;\Theta))\right\|_2^2\ , \tag{6}$$

where $\mathbf{w}_T$ contains the final approximation of the weights since T is the last layer of our network, and the vector Θ stores all our network parameters ($\mathbf{D}_h$, $\mathbf{B}$, $\mathbf{S}_t$ and λ_t for $t{=}1...T$).

Sobolev synthesis error. However, using the ℓ_2 norm measure alone in the loss function is not sufficient to correctly differentiate high frequency structures. Thus, we also employ the ℓ_2 norm of the gradient error between synthesized patches and ground-truth patches:

$$\mathcal{E}_g = \sum_{i=1}^{K} \left\|\nabla[\mathbf{y}_h^i] - \nabla[\mathbf{y}_l^i + \mathbf{D}_h\mathbf{w}_T(\mathbf{y}_l^i;\Theta)]\right\|_2^2\ . \tag{7}$$

where $\nabla[\cdot]$ is a component-wise gradient operator defined as $\nabla[\mathbf{x}] = [\nabla x_1, \nabla x_2, ..., \nabla x_n]^T$.

Divergence synthesis error. Since we are synthesizing incompressible fluid flows, it also makes sense to include in the loss function a measure of the divergence error between synthesized and ground-truth patches. While ground-truth patches are divergence-free if a patch representation purely based on the local vector field is used, we will argue that other fields (e.g., vorticity) can be used as well; hence for generality, we use:

$$\mathcal{E}_d = \sum_{i=1}^{K} \left\|\nabla\cdot(\mathbf{y}_h^i) - \nabla\cdot(\mathbf{y}_l^i + \mathbf{D}_h\mathbf{w}_T(\mathbf{y}_l^i;\Theta))\right\|_2^2\ . \tag{8}$$

Final form of our loss function. We express our loss function as:

$$L_T(\Theta) = \alpha_l\mathcal{E}_l + \alpha_g\mathcal{E}_g + \alpha_d\mathcal{E}_d + \alpha_\Theta\,\|\Theta\|_2^2\ , \tag{9}$$

where the last ℓ_2 norm on the network parameters helps avoiding over-fitting during learning. The parameters α_l, α_g, α_d and α_Θ help balance between training and test losses, and we set them to $\alpha_l{=}1$, $\alpha_g{=}0.05$, $\alpha_d{=}0.05$ and $\alpha_\Theta{=}0.5$ in all our training experiments. One may notice that these values differ significantly, especially for the terms involving gradients; this is because although input velocities are normalized, the gradient values may have much larger ranges, which should be given smaller parameter values.

3.4 Augmented patch encoding

Until now, we have not discussed what is exactly encoded in a local patch. Since we are trying to upsample a vector field in order to visualize the fine behavior of a smoke sequence, an obvious encoding of the local coarse flow is to use a small patch of coarse velocities, storing the velocities from an $n_c{\times}n_c{\times}n_c$ neighborhood into a vector $\mathbf{y}_l$ of length $N{=}3n_c^3$ in this case; a high-resolution patch is similarly encoded, involving a finer subgrid of size $n_f{\times}n_f{\times}n_f$ representing the same or smaller spatial neighborhood as the coarse patch — to make sure the coarse patch serves as a good predictor for the fine one. Using our network with such an encoding already performs reasonably well as Fig. 5(e) demonstrates, but the results are not fully spatially and temporally coherent, at times creating visual artifacts. Fortunately, we designed our approach to be general so that a number of improvements can be made to remedy this situation. Of course, growing the size n of the local patch itself would be one solution, but it would come at the cost of a dramatic increase in computational complexity and learning time, defeating the very

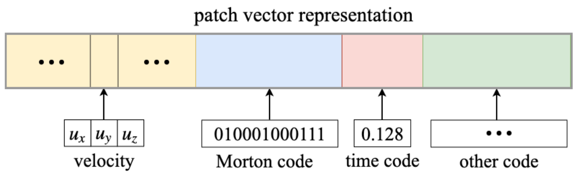


Fig. 7. **Augmented patch through space-time encoding.** We can use an augmented patch representation to improve spatial and temporal coherence: in addition to the velocity field, we add Morton code of the patch center, the time code of simulation time step the patch comes from, as well as any other relevant codes involved in the simulation, such as inlet size and position, to form our new patch representation vector.

purpose of our effort. We can, instead, keep the same spatial patch size n, but augment the patch with extra data to further improve spatio-temporal coherence by making the prediction of our residual dictionary less myopic: increasing N offers *more dynamic context* for both learning and synthesis of smoke flows.

Space-time encoding. For very contrived examples where there is no significant changes in the scene to upsample compared to the learning examples, we can add space and time encoding to the patch by augmenting each input patch vector with spatial and temporal components as sketched in Fig 7. To encode the patch position, Morton codes [Karras 2012] can be used as they have nice locality properties compared to a simple 3D offset vector. For each local patch, the Morton code corresponding to its center is simply added to the representation vector of that patch. For temporal encoding, the time step normalized by the maximum number of time steps of the simulation sequence can also be added to the representation vector. In addition, to support variation of flow conditions, the various simulation parameters (such as different inlet sizes and positions) can be taken as extra codes to be added to the representation vector. Knowledge of the position and time as well as the system parameters that a patch in the training set is coming from obviously guide the synthesis tremendously, as their relevance to a similar simulation is directly encoded into the patch representation. However, this brute-force encoding is *very* rigid, and should only be employed for scenarios where the animation sequence to be upsampled and simulation parameters are quite similar to the training simulations. Figs. 5(f), 19 & 20 show how exceedingly well the results of this approach can perform, providing a very efficient exploration through learning of motions near a given set of animation sequences.

Phase-space encoding. However, space-time encoding prevents more universal generality: if a simulation sequence to be upsampled is markedly different from any of the training simulation sequences, adding space and time information to the patches can in fact degrade the results as it implicitly guides the network to use patches in similar places and at similar times even if it is absolutely not appropriate in the new simulation. Instead, we wish to augment patch data with more information about the *local dynamical behavior of the flow*. One simple idea is to use phase space information: instead of using only the local vector field stored as $\mathbf{y}_l$, we can encode the patch with the time history of this local patch: $[\mathbf{y}_l^t, \mathbf{y}_l^{t-1}, ..., \mathbf{y}_l^{t-\tau}]$ where τ is the maximum number of previous time steps to use. Note that just picking $\tau{=}2$ corresponds in fact to the typical input of a full-blown integrator: knowing both the current and previous local vector fields is enough to know both velocity and acceleration of the flow locally. Our upsampling approach using this $\tau{=}2$ case can thus really be understood as a learned predictor-corrector integrator of the fine motion based on the two previous coarse motions: the coarse simulation serves, once directly upsampled to a higher-resolution grid, as a prediction, to which a correction is added via learned dynamic behaviors from coarse-fine animation pairs. Figs. 1(a) & 17 show the results of such a phase-space representation, with which a variety of synthesis results can be obtained.

Comparing the results of the new patch encoding with the one containing only the velocity field in Fig. 5(e), we see that the augmented representation captures much improved coherent vortical structures without obvious noise. While the synthesis results using a phase-space encoding may be slightly worse than the space-time encoded ones in terms of capturing the small-scale vortical structures of the corresponding high-resolution simulations, this significantly more general encoding can handle much larger differences (such as translations of inlets, rotations of obstacles, or even longer simulations than the training examples) in animation inputs.

Vorticity. For flows in general and smoke in particular, the visual saliency of vorticity is well known. Unsurprisingly, we found beneficial to also add the local vorticity field to the patch encoding: while this field is technically just a linear operator applied to the vector field, providing this extra information led to improved visual results, without hurting the learning rate. Consequently, and except for the few figures where space-time encoding is demonstrated, we always use only the last three vector fields and last three vorticity fields as the patch encoding, i.e., $[\mathbf{y}_l^t, \mathbf{y}_l^{t-1}, \mathbf{y}_l^{t-2}, \nabla\times\mathbf{y}_l^t, \nabla\times\mathbf{y}_l^{t-1}, \nabla\times\mathbf{y}_l^{t-2}]$.

Rotation. When synthesizing general flows, the overall flow field may be rotated compared to the training examples, e.g., when a coarse flow with an inlet is rotated by 90 degrees or when an obstacle is rotated by 45 degrees. In such a case, training from a set without this rotation may not lead to accurate results due to a lack of smoke motion in the proper direction (remember that we synthesize velocity fields, rather than density fields). To tackle this problem, we simply add rotated versions of each local patch to the training set. Several rotation angles can be sampled, for instance, each $\pi/2$ rotation for each coordinate direction. Fig. 17 shows a result using such a phase-space patch encoding including $\pi/2$-rotations, with coarse simulations containing obstacles that are rotated by 45 degrees in (e) & (f) along different coordinate directions. Obviously, these rotated versions of local patches are optional as they should not be included if the training simulations are clearly direction dependent, like in the case of gravity-driven flows. Fig. 8 summarizes the overall workflow for synthesizing high-resolution flow fields with our new network and augmented patch encoding.

3.5 Network learning

The neural network we just described can be trained by providing a large number of training pairs of coarse and fine simulation patches (we will discuss how to judiciously select candidate patches from a set of coarse and fine animation pairs in a later section): the loss function $L_T(\Theta)$ has to be minimized with respect to all the parameters stored in Θ, for instance, by the "Adam" method [Kingma and Ba 2014], with full-parameter update during optimization. However,

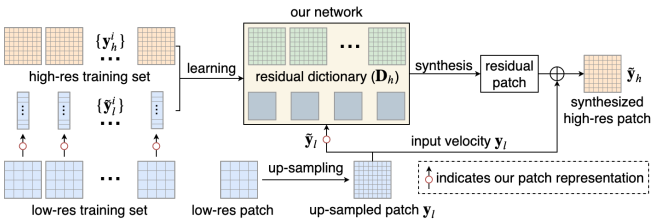


Fig. 8. **Our network-based dictionary learning approach.** In order to synthesize high-resolution flow fields, we first prepare a training set of local patch pairs ($\mathbf{y}_l^i$ and $\mathbf{y}_h^i$) from low- and high-resolution flow simulations respectively (left); note that the low resolution patches are represented by our augmented patch vector. With this training data, we learn a residual dictionary $\mathbf{D}_h$ as well as its associated predictor $\mathbf{w}_T(\mathbf{y}_l)$. Given a low resolution flow field, each local patch is fed to the network to predict a set of sparse coefficient $\mathbf{w}$ such that the high resolution patch can be synthesized using $\mathbf{D}_h$ and $\mathbf{w}$ added to the upsampled input patch.

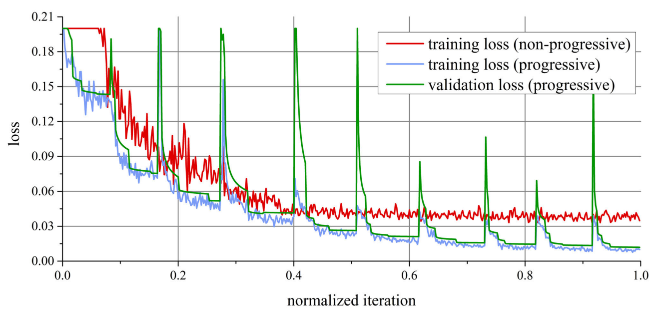


Fig. 9. **Training convergence.** Progressive vs. full-parameter (non-progressive) training exhibits different convergence rates, thus resulting in different training times and prediction accuracy.

a large number of layers and parameters may not produce good training convergence if a large variety of motions are present in the training set. In order to improve convergence — and thus, induce better prediction results during synthesis — we can employ a *progressive learning algorithm* similar to [Borgerding et al. 2017] which performs learning optimization in a cascading way, using the learned result from the previous layer as part of the initialization for the learning of the next layer. As the learning of one single layer involves only a small fully-connected network, and because this cascading approach to learning gradually provides better initialization than the full-parameter learning, this learning process turns out to exhibit better convergence.

ALGORITHM 1: Pseudo-code of our progressive learning algorithm.

Set up a parameter set Θ with $\mathbf{D}_h$, $\mathbf{B}$, $\mathbf{S}_1$, λ_1 and λ_{T+1}.
Initialize Θ with random numbers.
For (i= 1; i<T; i++) // T is the maximum number of layers
  Learn Θ for layer i to obtain $\mathbf{D}_h$, $\mathbf{B}$, λ_{T+1}, $\{\mathbf{S}_j\}$ and $\{\lambda_j\}$, $j = 1, ..., i$.
  Add $\mathbf{S}_{i+1}$ and λ_{i+1} into the parameter set Θ .
  Initialize $\mathbf{S}_{i+1}$ and λ_{i+1} with random numbers .
End For
Output learned parameters $\mathbf{D}_h$, $\mathbf{B}$, λ_{T+1}, $\{\mathbf{S}_i\}$ and $\{\lambda_i\}$, $i = 1, ..., T$.

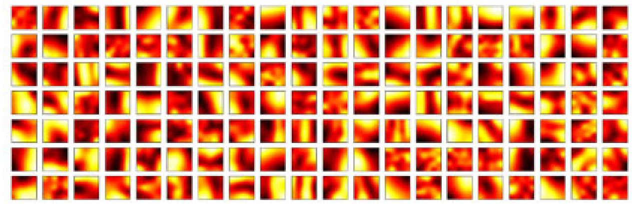


Fig. 10. **Example of learned dictionary.** Visualization of cross-sections of 3D velocity patches from a portion of the dictionary set.

More specifically, we first initialize all variables randomly in $[-0.01, 0.01]$ and perform learning for the first layer to find the optimal parameters $\mathbf{D}_h$, $\mathbf{B}$, $\mathbf{S}_1$ and λ_1. We then use these parameters as initialization for the learning phase of the second layer, where now another set of parameters $\mathbf{S}_2$ and λ_2 are added (with random initial values), and this new learning results in another set of optimal parameters for all the variables involved. This process repeats by adding $\mathbf{S}_{i+1}$ and λ_{i+1} into the learning for the $(i+1)$-th layer, with all other parameters from previous layers initialized to the learning result of the i-the layer, until all the layers in the network are learned. For each learning phase, we also employ the "Adam" method [Kingma and Ba 2014]. When using space-time encoding, we use 90% of the training patches for learning and the remaining 10% for validation; when phase-space encoding is used, we found preferable to use training patches from several simulation examples, and use patches from different simulation examples for validation to better test the generalization properties of the training. We obtain the final learning result once we reached the T-th (final) layer of the network. Alg. 1 illustrates the pseudo-code for our progressive learning process with better convergence properties.

We show in Fig. 9 the evolution of the loss function during a progressive (blue) vs. a non-progressive full-parameter (red) training, as well as the corresponding validation loss (green) during a typical training of our network. The periodic large peaks in the progressive learning curve indicate a transition from one layer to the next, where a subset of the randomly initialized values are inserted into the learning parameters, thus increasing the loss to a very high value; however, the loss quickly goes back down to an even smaller value due to better initialization. Compared to full-parameter learning, progressive learning systematically converges to a smaller loss for both training and validation sets, thus enabling better synthesis results.

(a) (b) (c)

Fig. 11. **Original *vs.* multiscale synthesis.** From training simulations only containing one inlet on the left of the domain, simulating a bottom inlet produces an adequate, but inaccurate upsampling (a); the same simulation using our multiscale network (b) produces a result much closer to the corresponding fine simulation (c).

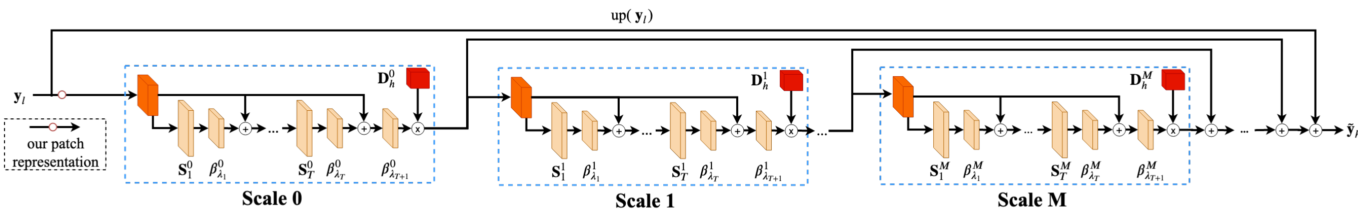


Fig. 12. **Multiscale network.** To increase the network representability, a multiscale version of our network can be employed. This network structure subdivides the residual patch into M multiple scales, and each scale is represented and learned by our original network. The synthesis result is obtained by summing together all the components that each of these subnetworks synthesizes.

At the end of either full-parameter or progressive learning, we obtain all network parameters, including the dictionaries. Fig. 10 shows a partial visualization of the learned dictionaries through small cross-sections of selected patches. It should be noted that, although progressive learning can produce better convergence (and thus better synthesis results), it can also end up being slower than full-parameter learning for very large training sets. In practice, we compromise between learning accuracy and efficiency: we use full-parameter learning for cases where the diversity of the training set is relatively large, and progressive learning otherwise (see Tab. 1).

3.6 Multiscale network

If our training set has very high diversity, the design of our network described so far may no longer be appropriate as shown in Fig.11(a) when rotated patches are added for training: if the training set contains too diverse a set of physical behaviors, $\mathbf{D}_h$ becomes too complex and can *exceed* the representability of the network. We could further increase the depth of the network and the size of the dictionary to increase the network capacity to handle more complex representations, but at the expense of significantly increased training and synthesis times. Instead, motivated by multiresolution analysis, we decompose $\mathbf{D}_h$ into multiple scales (components): $\mathbf{D}_h = \mathbf{D}_h^0 + \mathbf{D}_h^1 + ... + \mathbf{D}_h^M$, where each scale is represented by our previous LISTA-like network, resulting in a multiscale network as depicted in Fig. 12. Even if each sub-network is rather simple in its number of layers and dictionary size (and thus limited in its complexity of representation), the cumulative complexity of the resulting M-scale network is significantly increased. While the learning phase of this multiscale network could still follow the same progressive optimization process as we described above, we found it relatively slow to converge compared to a full-parameter optimization, for only a marginal gain in final loss. Thus, all of our examples based on a multiscale network (i.e., Figs. 1(a-c), 11, 16, 17 and 18) were trained via full-parameter optimization, with $M = 2$ since a two-level hierarchy proved sufficient in practice. Fig. 11(b) shows the synthesis from such a multiscale network when rotated patches are added to the training set, indicating that much better results can be obtained with this multiscale extension when compared to the corresponding fine physical simulation shown in Fig.11(c).

3.7 Assembly of training set

For network training, we need to prepare a large number of training pairs of corresponding low-resolution and high-resolution patches. The patch size should be carefully chosen to tune efficiency and visual coherence. Too small a size may not capture sufficient structures, whereas too large a size may require a very large dictionary and thus slower training and more non-zero coefficients during synthesis, hampering the overall computational efficiency. In practice, we found that a low-resolution patch size of $n_c = 3$ and a high-resolution patch size of $n_f = 5$ offer a good compromise, and *all our results were generated with these patch sizes.* In general, these small patches should come from a set of different simulation sequences with different boundary conditions, obstacles, or physical parameters to offer enough diversity for our training approach to learn from. The training patch pairs are then automatically selected from these sequences. Instead of using the entire set of local patches from all sequences, we found that proper patch sub-selection is important for efficiency: getting a good diversity of patches gives better guidance for the network training and higher convergence, thus producing better synthesis results. We thus employ importance sampling (as used in other learning-based fluid simulation work, e.g., [Kim and Delaney 2013]), where the patch selection is done using the numpy library [Oliphant 2006] for a probability distribution based on the vorticity of the flow field and the smoke density of smoke (i.e., the local number of passive tracers advected in the flow to visualize the smoke) on either low- or high-resolution simulations: in essence, we favor regions where smoke is likely to be or to accumulate during an animation to better learn what is visually most relevant. Another criterion of visual importance that we found interesting to leverage during patch selection is a large local strain rate: since turbulent flows are particularly interesting due to their small scale structures, targeting predominantly these regions where wisps of smoke are likely to be present allows the network to better synthesize these salient features. Fig. 13 shows an illustration of such an importance sampling, where color luminosity indicates sampling importance.

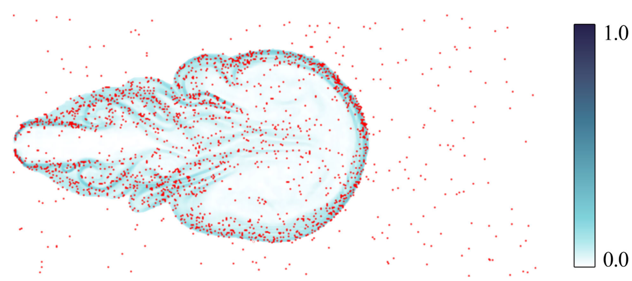


Fig. 13. **Importance sampling for training.** We select our training patches based on an importance sampling calculated from smoke density and local strain, where darker colors indicate higher importance (hence more selected patches); red dots show selected training patch centers.

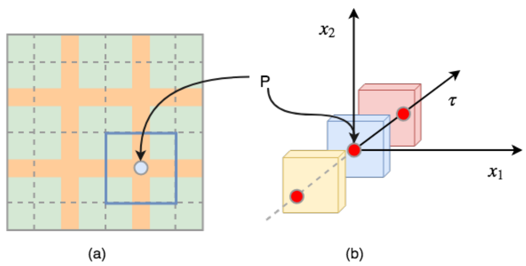


Fig. 14. **Patch blending.** 2D illustration of our 4D convolution of overlapped patches: (a) patches are laid out with overlapping (regions in orange); (b) for each node in the overlapped region, the overlapping space is shown along the τ direction, where the center of coordinate in that space is placed at the patch with no overlap, see the dotted line in (a).

3.8 High-resolution flow synthesis

After learning, the network automatically predicts high-resolution patches from low-resolution input ones by evaluating the local sparse weights that best reconstruct the dynamical surrounding of each patch. To further improve spatial coherency, we evaluate overlapping high-resolution patches, then blending of the overlapped regions (see orange regions in Fig. 14(a) as an example) is performed (in parallel) to ensure a smooth global reconstruction. Different blending approaches could be used; we settled on a convolution-based method as follows. We consider the synthesized velocity $\mathbf{u}(\mathbf{x}, \tau)$ in overlapped regions as a 4D function separately in 3D space ($\mathbf{x}$) and an overlapping space coordinate (τ) (see Fig. 14(b)), and employ a 4D Gaussian kernel $\mathrm{G}(\sigma_\mathbf{x}, \sigma_\tau)$ to do the convolution, with $\sigma_\mathbf{x}$ and σ_τ the standard deviations for spatial and overlapping domains, respectively. We set $\sigma_\mathbf{x} = 2.5$ and $\sigma_\tau = 1.5$ in all our experiments. Since the whole convolution is separable, it can be formulated as:

$$\mathbf{u}(\mathbf{x}, \tau) \leftarrow \mathrm{G}(\sigma_\tau) * [\mathrm{G}(\sigma_\mathbf{x}) * \mathbf{u}(\mathbf{x}, \tau)] \,. \tag{10}$$

This means that we first conduct a 3D convolution in the spatial domain followed by a 1D convolution in overlapping space after local patch prediction to obtain the final synthesized result for the whole high-resolution field.

4 RESULTS

We now discuss the various results presented in this paper. Most of the datasets used for training the network and synthesizing our results were collected from the kinetic fluid simulation method of [Li et al. 2019], except for the coupling example in Fig. 16(c) where the recent kinetic approach of [Li et al. 2020] was used. However, our method is not restricted to a specific fluid solver: we can start from an arbitrary set of time-varying vector fields simulating a given physical phenomenon.

4.1 Implementation Details

While our approach can handle basically any dictionary or patch size, we first discuss the different choices of implementation parameters we used in our examples for reproducibility.

Training details. In our implementation, we combine all patches that were collected for training to form a large matrix as input. The learning process is then achieved by a series of matrix products, which are evaluated in parallel by CUDA with the CUBLAS library [Nvidia 2008]. During learning, since we need to compute the

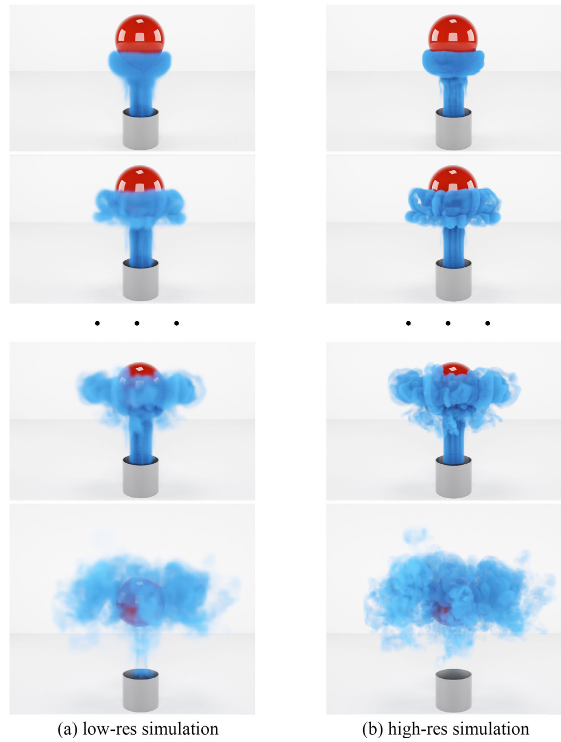


Fig. 15. **Sphere-in-air-jet training set for super-resolution**. From a high-resolution simulation of a vertical jet flow hitting a sphere (b), a low-resolution simulation is obtained through direct downsampling (a) to form a training pair of simulation for super-resolution.

gradient tensor which is extremely large, we sample 4096 patches for its computation. The learning rate l_r involved in the parameter matrices is dynamically changed: initially, it is given a relatively large value, e.g., $l_r = 0.0001$; as iterations converge with this fixed l_r, we further decrease it until final convergence, i.e., when further reducing l_r does not change the loss anymore.

Flow synthesis. The synthesis process is also implemented by a series of matrix products in parallel. We first collect all overlapped local patches to form a large matrix as input, and then go through the network by a series of parallel matrix calculations for synthesizing the high-resolution patches. A parallel convolution in overlapped regions is finally performed to obtain the synthesized high-resolution field, in which passive tracers can then be advected to render the smoke. Depending on the time resolution of the coarse inputs, we only upsample a fraction (between a third and a tenth) of the coarse simulation time steps; this is usually enough to create upsampled high-resolution flow fields that are then used to advect smoke particles or high-resolution density fields, and render the animation.

Libraries and memory requirements. Our learning was implemented with TensorFlow [Abadi et al. 2016] on a server using NVIDIA P40 GPUs, each with a total memory of 24GB. For large training set (larger in size than the allowable GPU memory), we perform out-of-core computing by evaluating the loss and gradient with several passes and data loads. The synthesis process was implemented on a basic workstation equipped with an NVIDIA GeForce RTX 2080 Ti

Table 1. **Statistics.** We provide in this table the parameters, timings and memory use per frame for various smoke animations shown in this paper.

Items	Figs. 1(a)&16(a)	Fig. 16(b)	Fig. 16(c)	Fig. 16(d)	Figs. 1(b)&17	Fig. 1(c)&18	Fig. 1(d)&20	Fig. 19	Fig. 24(d)	Fig. 24(e)	Fig. 24(f)
Resolution (coarse)	60×60×60	50×50×50	100×100×60	200×50×50	50×50×50	50×50×50	100×50×100	30×90×90	25×37×25	25×37×25	25×37×25
Resolution (fine)	240×240×240	200×200×200	400×400×240	800×200×200	200×200×200	200×200×200	400×200×400	120×360×360	50×75×50	100×150×100	200×300×200
Network structurea	multiscale	multiscale	multiscale	multiscale	multiscale	multiscale	single scale	single scale	single scale	single scale	single scale
Learning method	full-parameter	full-parameter	full-parameter	full-parameter	full-parameter	full-parameter	progressive	progressive	progressive	progressive	progressive
Patch encoding method	phase-space	phase-space	phase-space	phase-space	phase-space	phase-space	space-time	space-time	space-time	space-time	space-time
Dictionary size	800	800	800	800	800	800	800	800	400	400	400
Network memory size	49M	49M	49M	49M	49M	49M	22M	22M	4M	8M	9M
Training setup time	10 hours	10 hours	10 hours	10 hours	16 hours	20 hours	8 hours	6 hours	2 hours	3 hours	5 hours
Training time	18 hours	18 hours	18 hours	18 hours	72 hours	65 hours	37 hours	31 hours	15 hours	19 hours	27 hours
Time cost (coarse)	0.028 sec.	0.016 sec.	0.51 sec.	0.097 sec.	0.016 sec.	0.016 sec.	0.098 sec.	0.031 sec.	0.0032 sec.	0.0032 sec.	0.0032 sec.
Time cost (fine)	5.92 sec.	2.87 sec.	112.8 sec.	23.62 sec.	2.87 sec.	2.87 sec.	23.6 sec.	6.84 sec.	0.025 sec.	0.33 sec.	5.15 sec.
Time (upsampling)	1.28 sec.	0.74 sec.	3.69 sec.	3.1 sec.	0.74 sec.	0.74 sec.	1.05 sec.	0.49 sec.	0.013 sec.	0.044 sec.	0.35 sec.
Speed-up	4.6	3.9	30.6	7.6	3.9	3.9	22.5	13.9	1.9	7.5	14.7

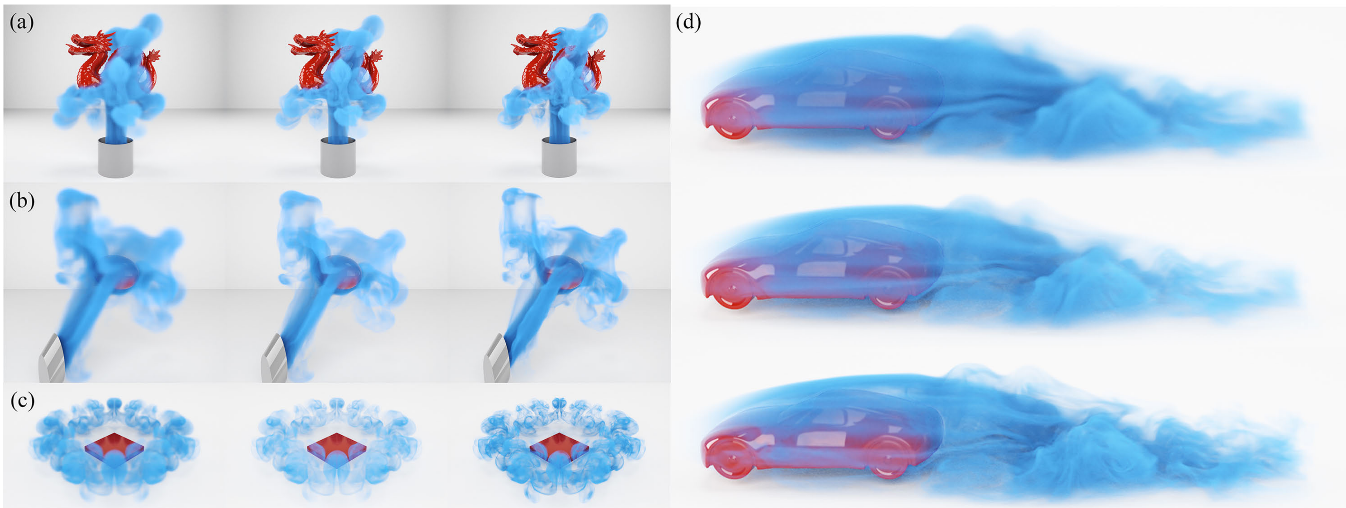


Fig. 16. **Super-resolution**. From a single training simulation of a sphere in a jet flow as shown in Fig. 15 (where the coarse simulation is a downsampled of a high-resolution simulation), we can synthesize with phase-space encoding a large variety of flow simulations: (a) a vertical jet flow through a dragon-shaped obstacle; (b) a jet flow from a tilted inlet hitting an ellipsoid; (c) turbulent smoke induced by the fall of a plate on the floor; or (d) a wind-tunnel simulation of a car. In each example, we show the low-resolution input, the synthesized high-resolution result from tempoGAN network [Xie et al. 2018], and our synthesized result respectively. Our approach captures visually crisper flow details than tempoGAN in all these cases.

GPU with 12GB memory. The final rendering is achieved with a particle smoke renderer [Zhang et al. 2015] together with the NVIDIA OptiX ray-tracing engine [Parker et al. 2010], which usually takes about 40 seconds to render one frame of animation for a resolution of 1280×720 as the output image with multisample anti-aliasing (3×3 samples per each output pixel), and with a maximum number of particles equal to 15M. After learning, the whole network (including the resulting dictionary) takes up a total size of approximately 50MB for the generalized synthesis case (Fig. 17), which has the highest amount of memory consumption among our learning results shown in this paper. Tab. 1 lists the network size and additional statistics of other synthesis cases.

4.2 Examples for various usage scenarios

Compared to existing learning-based methods, our proposed approach offers a more general framework for synthesizing high-resolution simulations of smoke flows from coarse simulations. We demonstrate its generality by reviewing the four different scenarios that our algorithm can handle based on the choice of training sets and inputs: *super-resolution* (when a downsampled animation is provided as an input), *generalized upsampling* (when the input is a coarse animation differing significantly from the training set), *restricted upsampling* (when the coarse input is close to a restricted set of training simulations), and *re-simulation* (when the coarse input is a small alteration of the simulation used for training). These four scenarios lead to different generalization behaviors and synthesis accuracy as we now detail.

Super-resolution. The first application of our dynamic upsampling of smoke flows is what Xie et al. [2018] called super-resolution, where a spatially-downsampled animation needs to be upsampled; this scenario allows for the efficient reconstruction of high-resolution animations that have been compressed through downsampling. In this particular context, training pairs are assembled from high-resolution simulations and their downsampled versions, see Fig. 15 for an example. Since such a training set ensures that the overall flow structures between the low-resolution and high-solution flow fields always match, our neural network easily learns how to derive high-resolution details from a coarse, downsampled animation, even for turbulent flows where the flow structures tend to be chaotic. Fig. 16 demonstrates a variety of smoke animation results synthesized from low-resolution simulation inputs, which all purposely differ significantly from the single training pair shown in Fig. 15; compared with

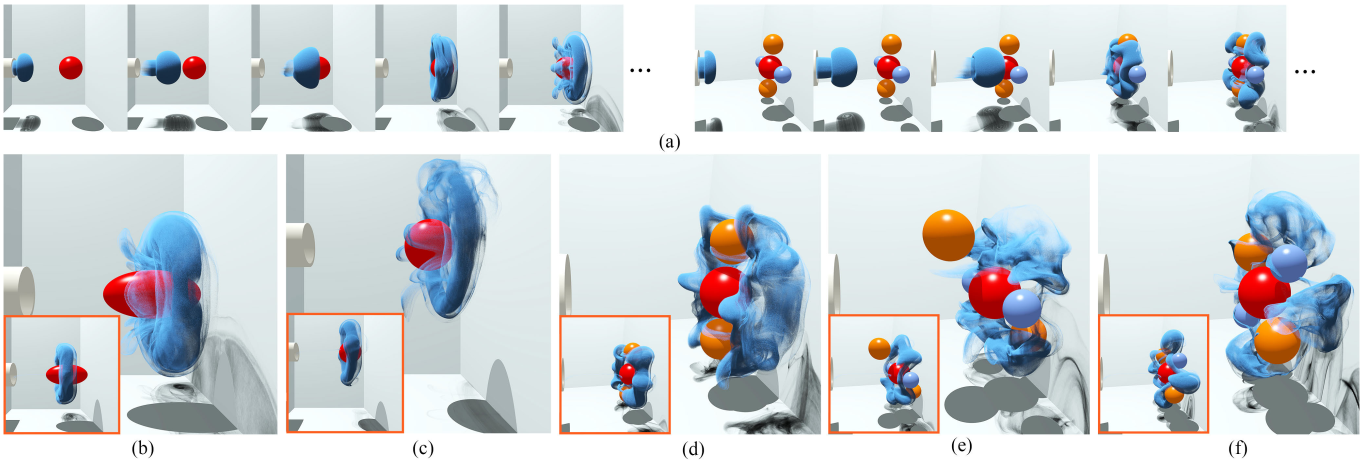


Fig. 17. **Generalized upsampling.** From only two simulation examples containing different numbers of sphere obstacles (a), our network-based approach can upsample coarse simulations (with phase-space encoding and an expansion ratio of 4×4×4=64) for different obstacle shapes (b), different inlet positions and longer simulation time (c), different arrangement of obstacles than the training set (d), and different inclinations of the obstacles (e) & (f), to show the generalizability of our network.

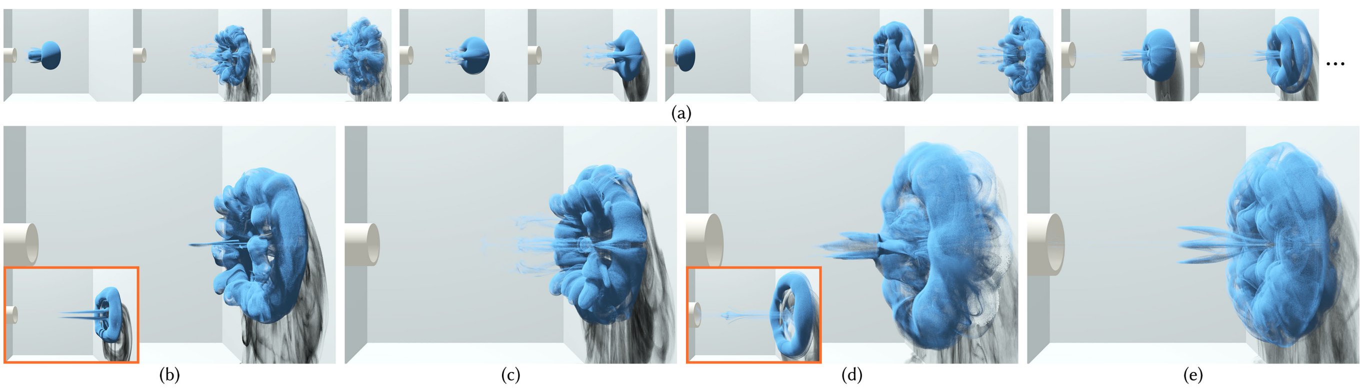
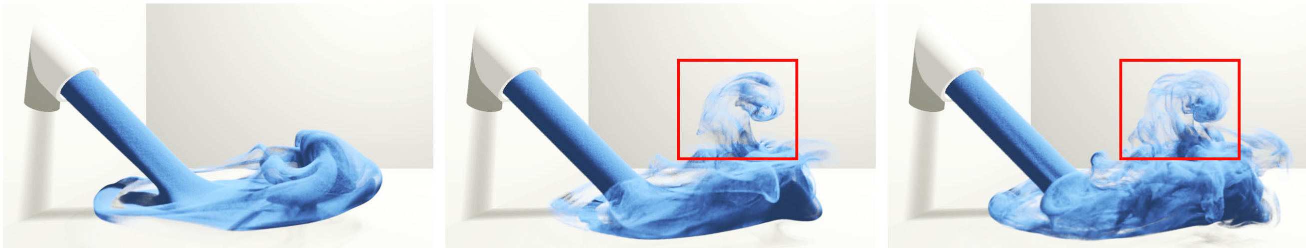


Fig. 18. **Restricted upsampling.** From a series of input coarse/fine animation sequences with only changes of the inlet size (a), our network-based approach can upsample smoke simulations (with phase-space encoding and an expansion ratio of 4×4×4=64) with arbitrary inlet sizes in between those used in the training set: (b) & (d) for two different inlet sizes not present in the training set. Compared to the corresponding ground-truth numerical simulations (c) & (e), our synthesized results share close resemblance.

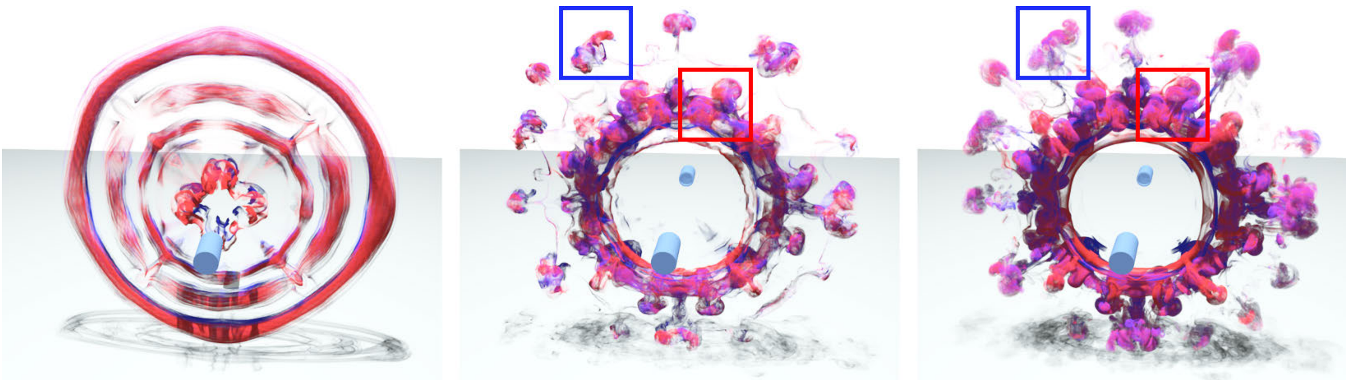
tempoGAN [Xie et al. 2018] (we directly used their trained network parameters), our approach generates finer smoke plumes as well as other visually-obvious flow structures. Note however that this scenario does not ensure that the synthesized high-resolution flow field can match their corresponding physical simulations: it is only intended to generate visually plausible smoke results with much finer details than in the input.

Generalized upsampling. Arguably the most challenging task is to generate a plausible high-resolution smoke flow from a coarse input that shares very little in common with the training set. This is what we call generalized upsampling, for which the training pairs are corresponding low- and high-resolution flow fields that are both physically simulated. Intuitively, upsampling in this case can only add physically-accurate details to a coarse simulated input if learning uses a very large set of patches from a variety of training simulations so as to cover a sufficient variety of simulation conditions. While guaranteeing physical accuracy is not possible in this context, we demonstrate in Fig. 17 that even only two training simulations (with respectively one and five spheres) are enough to train our dictionary-based upsampling process to deal with a fairly large parameter space (inlet position and diameter, obstacle position and shape, size and orientation, etc.) to result in plausible upsampling: the resulting neural network can handle different coarse simulations, including changing the shape of the sphere, shifting the inlet position and increasing the simulation duration, removing some of the sphere obstacles, as well as rotating the sphere obstacles by 45 degrees (this configuration is also not present in the training set since only 90-degree patch rotations were added). Although the synthesized high-resolution simulations do not match their corresponding fine physical simulations closely due to the typical chaotic



(a) (b) (c)

Fig. 19. **Smoke shooting.** From a low-resolution (30×90×90) simulation input (a), our synthesized smoke (b) with space-time encoding at high resolution (120×360×360), vs. the high-resolution simulation (c) for reference. Despite a factor of 64 (4×4×4) in resolution ratio, visually important structures (e.g., the secondary vortex ring marked with a red box) which were not in the coarse simulation (a) but present in the fine simulation are well captured.



(a) (b) (c)

Fig. 20. **Vortex rings colliding.** From a low-resolution (100×50×100) simulation input (a), our synthesized smoke (b) with space-time encoding at high resolution (400×200×400), vs. the high-resolution simulation (c) for reference. Despite a factor of 64 (4×4×4) in resolution ratio, we can still faithfully capture the obviously important vortex structures (e.g., the first (red box) and secondary (blue box) vortices, with leapfrogging in the center) present in the fine simulation.

behavior of smoke flows (Fig. 2), our trained network generates plausible high-resolution vortex structures far better than if only noise- or high-frequency structures were added to the coarse simulations. This illustrates the power of our approach: just a few training simulations can serve as a decent learning catalog to upsample coarse simulations. Training with a larger set of simulation pairs creates a network that can handle inputs deviating even more significantly from the training set, at the cost of more computational resources.

Restricted upsampling. If training simulations and inputs are less varied, our approach offers better training and more accurate synthesis. For example, for the jet flow smoke shown in Fig. 18, we collect training patches from simulations using only four different inlet sizes, with phase-space encoding and with inlet size as an additional patch code, to synthesize high-resolution simulation results from a coarse simulation with an arbitrary inlet size different from those used in the training set. The largest inlet in the training set is nearly twice as large as the smallest one, with two additional inlet sizes in between them to produce a total of four simulation sequences, from which training patches are sampled. Because of this more restricted setup, the synthesized high-resolution flows contain vortex structures that closely resemble the real fine simulations as shown in Figs. 18(c) & (e)). Similar restricted cases where we change, e.g., the obstacles' position or size can be performed as well.

Re-simulation. An extreme case of restricted upsampling, in which training patches are sampled from a *single* simulation and the input is a coarse simulation close to this training simulation with only small adjustments on initial and/or boundary conditions (see Figs. 19 & 20), amounts to re-simulation. This case is much narrower in its applicability for upsampling, but can produce near-perfect synthesis results, achieving simulations that are very close in the vortical structures to their corresponding physical simulations; see the secondary vortices in Figs. 19 & 20 for instance. The synthesis accuracy depends of course on the underlying Reynolds number, though: the lower the Reynolds number, the closer the synthesized simulation to its fine simulation counterpart.

5 DISCUSSION

Finally, we discuss a few important aspects of our learning approach to provide additional insight on its strengths and limitations.

5.1 Learning parameters

In our approach, the dictionary size can be arbitrarily set, with larger sizes providing more physically-plausible results but slower training. In practice, we set it to 400 for re-simulation cases shown in Figs. 24(d), (e) & (f); for other cases, we use 800 as listed in Tab. 1. As a rule of thumb, we recommend larger values for higher Reynolds

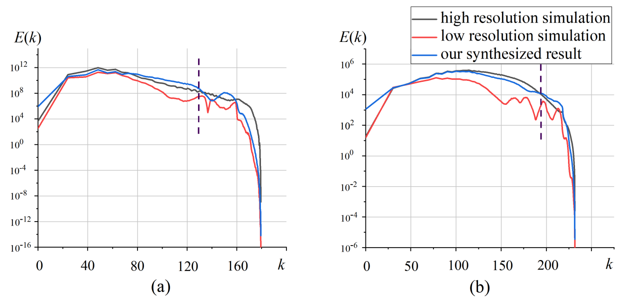


Fig. 21. **Spectral behavior.** We plot the energy spectra for a low-resolution simulation (red), a high-resolution simulation (gray) and our synthesized flow (blue) with respect to wavenumber k, for two types of upsampling scenarios: (a) generalized upsampling (Fig. 17(b)), and (b) re-simulation (Fig. 20). Our network produces spectra faithfully close to the high-resolution simulation counterparts below a critical wavenumber (dotted lines).

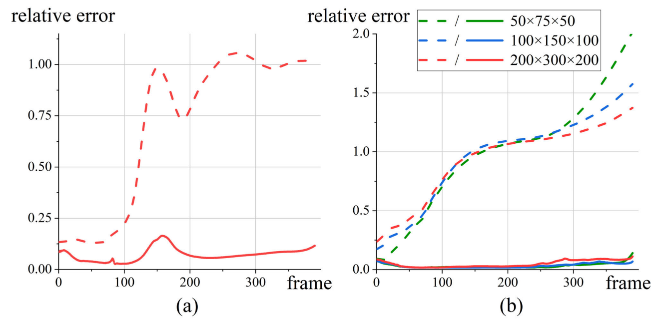


Fig. 22. **Synthesis error over time steps.** In (a), we plot relative L_2 errors for the generalized synthesis result from Fig. 17(b): the solid curve represents the error between our synthesized flow and the high-resolution simulated flows, while the dash-dotted curve shows the error between the coarse and fine simulations. In (b), we plot the same error curves, this time for the upsampling of the low-resolution simulation input (25×37×25) used in Fig. 24, for three different upsampling factors.

numbers — and conversely, smaller values for lower Reynolds numbers — to adapt to the complexity of the flow. In addition, some other network parameters should be set: for re-simulation shown in Figs. 24(d), (e) & (f), the sizes of $\mathbf{B}$, $\mathbf{S}$ and $\mathbf{D}_h$ are 400×83, 400×400 and 400×375, respectively; for re-simulation shown in Figs. 19 & 20, the sizes of these parameters are 800×83, 800×800 and 800×375, respectively; for other upsampling, the sizes of these parameters are 800 × 486, 800 × 800, and 800 × 375, respectively. For generalized upsampling with large variation of flow conditions compared to the training simulations (see, e.g., Fig. 1(b) & Fig. 17), we used 22M patches (selected via importance sampling from 800 frames spread across different training simulations) to learn our dictionary and LISTA-like sparse coding layers. For more restricted upsampling cases where the variation of flow conditions is not significant, the number of training patches can be far lower: we used 15M for Fig. 1(c) & Fig. 18, 8M patches for Fig. 16, and 3M patches for Fig. 1(d) and Figs. 19 & 20.

5.2 Flow synthesis accuracy

As we highlighted early on, the technique proposed in this paper is not generally intended to produce high-resolution flow fields that are physically accurate. Since we target visually realistic smoke animations for relatively high Reynolds numbers, our method only ensures that plausible fine vortex structures are generated from the coarse input, without noticeable artifacts. There are several factors that affect the quality of our results. The two main parameters are the dictionary size and the number of network layers, which both influence the dimensionality of the space of synthesized patches. For flows with higher Reynolds numbers, one should use larger dictionary size since the local structures tend to be more complex. Another factor is how the training patches are sampled from the input coarse-fine animation pairs. In general, our method can capture most high-resolution flow structures, but our importance sampling may miss vortices that are only active for a very short period of time; therefore, our synthesis will not capture these very transient phenomena properly by lack of training. To a certain extent, the user may define a different notion of importance that highlights the most desirable features that synthesis is expected to recover. In addition, as discussed during the review of our results, the way the training set is prepared also influences synthesis accuracy; e.g., if the training set is prepared through downsampling like for the super-resolution case, our upsampling is unlikely to obtain a high-resolution simulation close to its fine numerical simulation from a coarse input, even if it generates visually plausible details.

Energy spectrum. One of the important measures of accuracy, particularly for turbulent flows, is the energy spectrum of the velocity field. Fig. 21 shows the spectral behavior for the generalized upsampling (a) and re-simulation (b) cases. Below a certain critical wavenumber (indicated via a dotted line), both spectra plots match the corresponding simulations well, indicating that both types of flow synthesis methods can retain large-scale vortex structures present in the high-resolution simulations; note that re-simulation has a higher critical wavenumber, meaning that it captures smaller-scale vortex structures, as expected.

Synthesis error over time steps. Another way to assess the accuracy of our synthesis result is to compute the mean squared error of velocity fields normalized with respect to the numerical simulation at the same high resolution. Fig. 22 plots the resulting error variations over different time steps for the generalized synthesis case, as well as for different resolution ratios. We also plot the error between

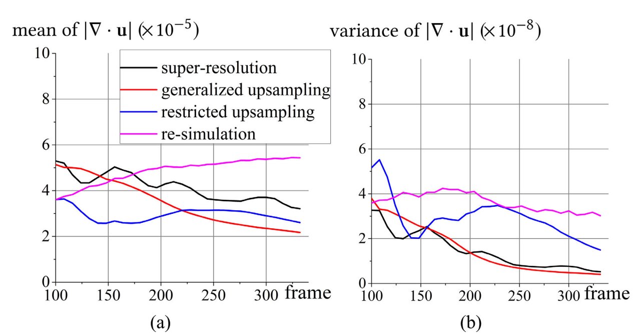


Fig. 23. **Incompressibility.** To numerically verify the incompressibility of our upsampling approach, we plot the mean (a) and variance (b) of the (absolute values of the) divergence of our synthesized velocity fields over time for super-resolution (Fig. 16(c)), generalized upsampling (Fig. 17(b)), restricted upsampling (Fig. 18(b)) and re-simulation (Fig. 20) results.

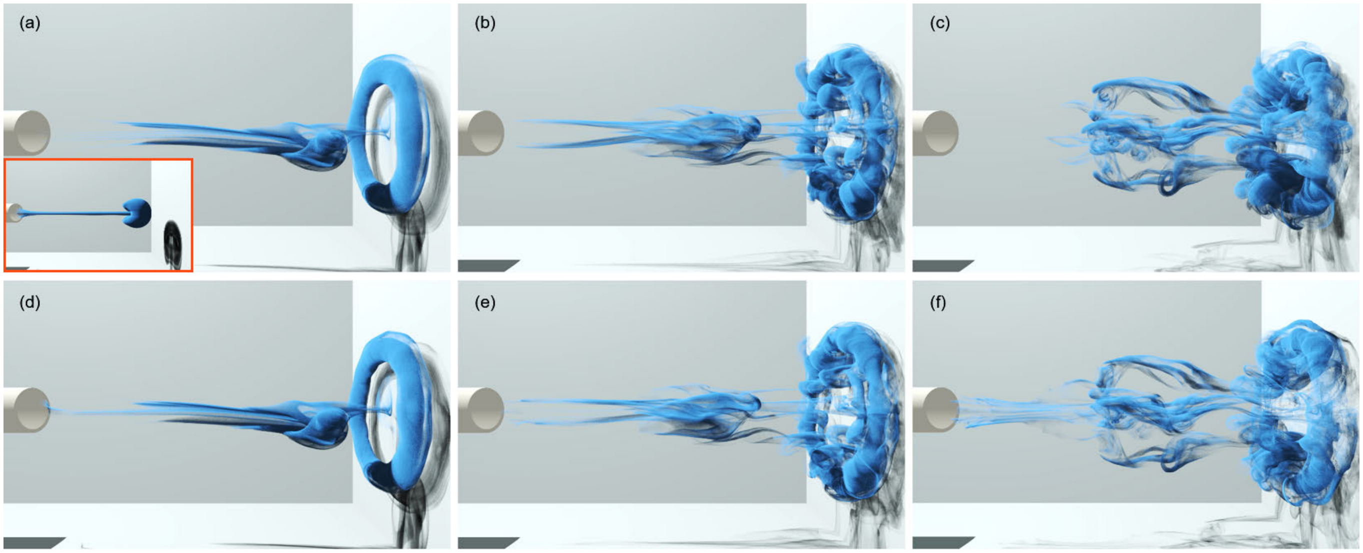


Fig. 24. **Different upsampling factors.** From the low resolution ($25 \times 37 \times 25$) smoke flow shown as an inset, the corresponding fine (top) and synthesized (bottom) animations (using space-time encoding) are shown at different resolutions: (a/d) $50 \times 75 \times 50$, (b/e) $100 \times 150 \times 100$ and (c/f) $200 \times 300 \times 200$.

coarse and fine simulations as a reference to better illustrate our synthesis accuracy. Our synthesis error remains relatively small and bounded over time for these test cases.

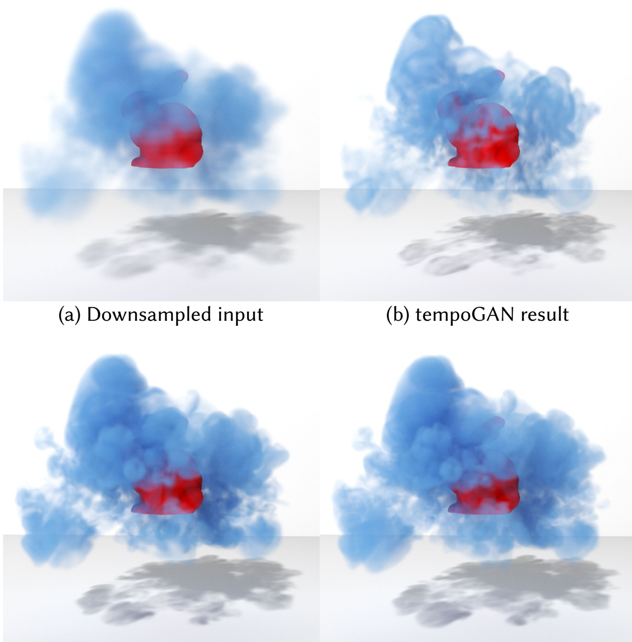
Incompressibility. Fig. 23 shows the mean and variance of the absolute values of the velocity divergence for a series of our upsampling results. As these values are uniformly very small for different types of flow synthesis, incompressibility is well preserved in practice.

Vortex structure preservation. While our approach can capture detailed vortex structures close to their fine simulation counterparts, our synthesis results sometimes exhibit crisper volutes, with less apparent diffusion in the rendered smoke compared to real physical simulations. Two reasons explain this behavior: first, physical simulations capture smaller-scale vortices which locally diffuse the smoke particles more than in the synthesized results; second, our network does not perfectly ensure spatial and temporal coherence among patches, so small mismatch between nearby patches can create local vorticity that attracts smoke particles rather than diffuse them. One could add a local diffusion to simply counteract this effect; we kept all our results *as is*, because a crisper look is, in fact, visually more attractive, and we also did not want to alter the results with post-processing in any way, which would obfuscate the interpretation of our results. It may also be noted that high-resolution structures are often synthesized even if the low-resolution simulation has seemingly not even any smoke in the area (see the trailing wisps in Fig. 24 or the rising plumes in Fig. 19): as we synthesize a high-resolution velocity field directly rather than smoke density, low-resolution flows can have small velocity variations in regions where no smoke particles were driven towards, but our network has learned that these velocity configurations become, in fact, full-blown smoke structures at high resolution.

Synthesis for different resolutions. While most of our results were using an expansion factor of $4^3 = 64$, we tried upsampling up to a ratio of 8 in each dimension, and obtained reasonable synthesis results as demonstrated in Fig. 24.

5.3 Generalizability

We showed in Sec. 4.2 that our approach can handle inputs quite different from the training set in the super-resolution and generalized upsampling scenarios: from a varied set of simulation patches, smoke flows can be quickly generated from a coarse input through "interpolation" and moderate "extrapolation" of the training patches to capture plausible fine physical structures in the flow. Restricted upsampling and re-simulation offer very limited generalizability as the training sets do not cover a large variety of examples; but this restriction also improves the physical accuracy of the resulting high-resolution flows. The examples we show in this paper were designed to showcase how our approach can generalize a smoke flow based on different training sets, without suffering from overfitting issues. The generalized upsampling in Fig. 17, for instance, relies only on *two* training simulations, one using a single sphere obstacle, and the second one using a set of 5 spheres placed on a common vertical plane. Synthesizing upsampled flows from only these two sequences with significant variations of the initial conditions (e.g., by either adding/removing sphere obstacles, changing the obstacle shape, or rotating the 5-sphere configuration by an angle) lead to visually plausible results. Super-resolution allows for even stronger generalizability as shown in Fig. 16: with only one simulation example of a jet flow through a sphere used for training, our network can be widely applied to a large variation of flow conditions, including a case with dynamic fluid-solid coupling, while still providing plausible results. Note that even the re-simulation examples in Figs. 19 and 20 exhibit some amount of both interpolation and extrapolation from the original simulation: recall that the training for re-simulation uses only *a small subset* of all patches, sampled over time and space; so a synthesized re-simulation relies on linear



(a) Downsampled input (b) tempoGAN result

(c) Our result (d) High-resolution animation

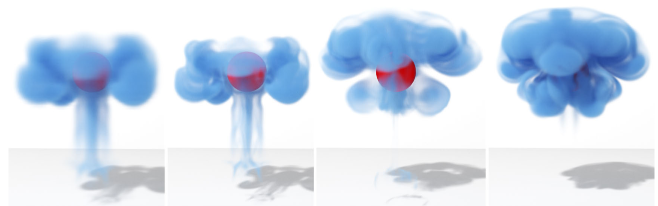
Fig. 25. **From downsampled to upsampled flows.** From a smoke animation computed from a downsampled (4 times along each dimension) fine animation, tempoGAN [Xie et al. 2018] can make the smoke look sharper (b), but fail to capture the correct dynamics of the fine simulation (d); our approach (using phase-space encoding), instead, captures the smoke animation much more closely.

combinations of these patches to synthesize the high-resolution flow, instead of directly replaying the fine animation. It is thus clear from our demonstrated results that our method can accommodate a large range of applications by varying the types of training sets used.

5.4 Comparison with other upsampling approaches

There are only a few previous works that can synthesize, based on a neural network, plausible high-resolution flow fields from low-resolution simulation inputs. The most relevant approach, proposed by Chu et al. [2017], used a CNN-based feature descriptor to synthesize high-resolution smoke details, also based on a local patch-based synthesis scheme. However, they relied on a nearest-neighbor search during synthesis, which greatly restricts the space of synthesized flow structures and makes the animation results often visually unnatural: smoke structures appear biased towards particular directions; see Fig. 12 in their paper for an example.

Another relevant recent work is the tempoGAN network of [Xie et al. 2018], which targets super-resolution. In addition to the visual comparisons we provided for super-resolution examples showing finer and crisper high-resolution simulations than tempoGAN, Fig. 25 shows yet another comparison with tempoGAN, this time with the corresponding high-resolution smoke animation result (Fig. 25(d)) provided as ground-truth. Our synthesized result is much closer to the ground-truth than tempoGAN, which remains too similar to the low-resolution flow. Note that we used the same training



(a) (b) (c) (d)

Fig. 26. **Comparison with tempoGAN network.** We perform upsampling comparison from a low-resolution numerical simulation input (50×50×50) between the tempoGAN network [Xie et al. 2018] and ours (using phase-space encoding), both at the resolution of 200×200×200: (a) low-resolution numerical simulation input; (b) tempoGAN upsampling result; (c) our network upsampling result; (d) the ground-truth fine numerical simulation.

sphere-in-air-jet example shown in Fig. 15 to synthesize the result in Fig. 25(c), while we used the trained network parameters of tempoGAN based on their own training sets. Moreover, our approach is also systematically faster to generate high-resolution flows: tempoGAN requires around 10 seconds to synthesize a flow at a resolution of 200×200×200, while ours takes merely 1 second for the same grid resolution and on the same GPU for fairness of evaluation. In addition, tempoGAN also takes significantly longer to train their network: while they need 9 days on this example, our method will only spend 18 hours. Finally, as discussed earlier, super-resolution is only one of the scenarios our approach can tackle: if generalized upsampling of a coarse simulation is needed, our learning approach still applies and performs well (see Fig. 26(c)&(d), for instance), while tempoGAN cannot handle this case.

5.5 Timings and effective compression

Regarding timings, training is slower for generalized upsampling due to the larger number of training patches used to allow for very varied inputs: the slowest training phase was 72 hours for Fig. 17 using TensorFlow with Nvidia P40 GPUs as described in Sec 4.1. For more restricted upsampling, it takes an average of approximately 50 hours to train our network: re-simulation typically requires 15 hours for training, while super-resolution requires 18 hours. Flow synthesis only takes around 1 second per time step of a high resolution simulation ($200 \times 200 \times 200$) from a low resolution input ($50 \times 50 \times 50$), rendering our approach much faster (often by approximately an order of magnitude or more) than the corresponding physical simulation; see Fig. 27 for performance speed-ups at various synthesis resolutions corresponding to Fig. 24. Note that we obtain our best speed-up factor for the coupling example of Fig. 16(c),since our upsampling does not suffer from time step restrictions for numerical stability compared to the actual fine simulation. Also note that what we refer as upsampling "speedup" as listed in Tab. 1 does not include computational times for density advection (or particle tracing) and training, since these stages are not formally part of the upsampling process. Memory usage is also significantly reduced with our approach: a high-resolution simulation typically requires from 1.6GB to 3.2GB for a resolution of 200×200×200 depending on the solver; instead, its low-resolution simulation only requires from 25MB to 50MB for a resolution of 50×50×50, and storing our whole

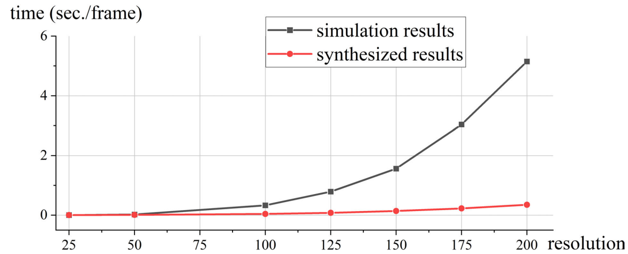


Fig. 27. **Speedup.** Comparison of performance under different resolutions for the upsampling example in Fig. 24 with restricted synthesis; gray curve indicates numerical simulation time at different resolutions, while red curve displays coarse simulation plus synthesis time for the same resolutions.

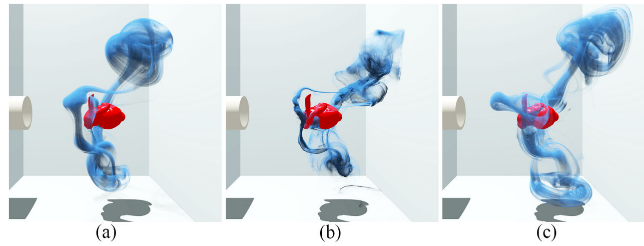


Fig. 28. **Potential limitations.** If we train our network using a flow over a sphere, upsampling a coarse animation of a turbulent flow over a bunny-shaped obstacle (a) can lead to significant inaccuracy (b) compared to the ground-truth solution (c): the training patches from the coarse simulation are simply not diverse enough to offer accurate prediction.

network requires a maximum of 50MB in the scenario requiring the largest training set (Fig. 17). Consequently, our learning-based upsampling approach can be considered as a very effective spatial and temporal compression scheme for both laminar and turbulent fluid flows. We leave a proper evaluation of its value to future work.

5.6 Limitations

Our method is not without limitations, however. One cannot expect poorly-chosen training sets to provide predictive upsampling as we now detail to help understand what to expect from our approach. First, our local patch approach cannot guarantee a perfect spatial and temporal coherence in the results — although visual artifacts are all but impossible to notice in practice. Note that the patch coherence is strongly related to the generalization property of the network. If a coarse input patch deviates significantly from the training patches, it will then be difficult to represent as a meaningful combination of training patches; the network behavior in this case is not quite predictable, and incoherence is likely to occur. This is particularly obvious when we prepare the training sets from coarse and fine simulation pairs with strong turbulence. In such a case, nearby mismatched patches may create large velocity gradients (along with a strong vorticity) in the overlapped regions, which will attract smoke particles and result in very thin and unnatural smoke features — see for instance Fig. 28, where we train our network with a simple flow simulation around a sphere (the training pairs are sampled from simulations in Fig. 26(a) & (d)), but synthesize a fast, turbulent flow around a bunny-shaped obstacle. More generally, very turbulent flows are simply difficult to upsample accurately: since they are chaotic, an arbitrary coarse simulation may contain patches widely different from even a large sample of training patches. Moreover, the difference between coarse and fine turbulent flows may increase exponentially over time, bringing an additional difficulty for such fast flows. However, if the coarse inputs are downsampled versions of fine simulations like it was assumed in tempoGAN [Xie et al. 2018] and also demonstrated in our super-resolution examples, inputs are of course much more "predictive" of the motion, even in the case of turbulent flows. Our approach outperforms tempoGAN in this specific case, both in terms of generalizability and efficiency. Second, since high-frequency components are synthesized by our network without a strict enforcement of divergence-freeness, it also does not guarantee incompressibility; however, given that the coarse simulation is already incompressible and that we enforce a divergence-free dictionary, the resulting high-resolution animation remains nearly incompressible. Lastly, our method may require a large amount of training patches to produce physically accurate results, which induces longer training time, especially for generalized upsampling. While our patch sampling was designed to keep the patch number low by promoting a diverse sampling of patch behaviors, our importance sampling strategy could be further refined to improve generalizability for a given count of training patches.

6 CONCLUSION

In this paper, we proposed a dictionary-based approach to synthesizing high-resolution flows from low-resolution numerical simulations for the efficient (possibly iterative) design of smoke animation. In sharp contrast to previous works that only add high-frequencies through noise or fast procedural models, our approach learns to efficiently predict the appearance of plausible fine details based on the results of coarse and fine offline numerical simulations. A novel multiscale dictionary learning neural network is formulated based on a space-time or phase-space encoding of the flow, and then trained through a set of coarse and fine pairs of animation sequences. From any input coarse simulation, a high-resolution simulation can then be approximated via a sparse representation of the local patches of the input simulation by simply applying our trained network per patch, followed by a sparse linear combination of high-resolution residual patches blended into a high-resolution grid of velocity vectors. We also highlighted the key advantages of our method with respect to previous methods that either just added high-frequency noise or used a very limited space of upsampled patches, and provided a clear analysis of the possible failure cases for fast and turbulent flows. We believe that our use of sparse combinations of patches from a well-chosen over-complete dictionary offers a rich basis for future neural-network based approaches to motion generation, not limited to smoke simulations.

7 ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their insightful comments. We would also like to thank Chaoyang Lyu, Yihui Ma, Yixin Chen, Xinghao Wang and Wenji Liu from FLARE Lab of ShanghaiTech University for helping with rendering and image/video editing. This work was supported by the Young Scientists Fund of the National Natural Science Foundation of China (Grant

No. 61502305) and the startup funding from ShanghaiTech University. Finally, Mathieu Desbrun acknowledges the hospitality of ShanghaiTech University during his sabbatical.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

Michal Aharon, Michael Elad, Alfred Bruckstein, et al. 2006. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing* 54, 11 (2006), 4311.

Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. 2012. Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics* 31, 4, Article 62 (July 2012), 8 pages.

Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics* 32, 4, Article 103 (July 2013), 10 pages.

Markus Becker and Matthias Teschner. 2007. Weakly compressible SPH for free surface flows. In *Symposium on Computer Animation*. 209–217.

Mark Borgerding, Philip Schniter, and Sundeep Rangan. 2017. Amp-inspired deep networks for sparse linear inverse problems. *IEEE Transactions on Signal Processing* 65, 16 (2017), 4293–4308.

Robert Bridson, Jim Houriham, and Marcus Nordenstam. 2007. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics* 26, 3, 46–es.

Tyson Brochu, Todd Keeler, and Robert Bridson. 2012. Linear-time Smoke Animation with Vortex Sheet Meshes. In *Symposium on Computer Animation*. 87–95.

Shiyi Chen and Gary D Doolen. 1998. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics* 30, 1 (1998), 329–364.

Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, and Steffen Weißmann. 2016. Schrödinger's smoke. *ACM Transactions on Graphics* 35, 4, Article 77 (July 2016), 13 pages.

Mengyu Chu and Nils Thuerey. 2017. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics* 36, 4, Article 69 (July 2017), 14 pages.

Ingrid Daubechies, Michel Defrise, and Christine De Mol. 2004. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics* 57, 11 (2004), 1413–1457.

Fernando de Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Transactions on Graphics* 34, 4, Article 50 (July 2015), 11 pages.

Alessandro De Rosis. 2017. Nonorthogonal central-moments-based lattice Boltzmann scheme in three dimensions. *Physical Review E* 95, 1 (2017), 013310.

Mathieu Desbrun and Marie-Paule Gascuel. 1996. Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies. In *Workshop on Computer Animation and Simulation*. 61–76.

Dominique d'Humières. 2002. Multiple–relaxation–time lattice Boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 360, 1792 (2002), 437–451.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual Simulation of Smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 15–22.

Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. 2017. A Polynomial Particle-in-cell Method. *ACM Transactions on Graphics* 36, 6, Article 222 (Nov. 2017), 12 pages.

Cristina Garcia-Cardona and Brendt Wohlberg. 2018. Convolutional dictionary learning: A comparative review and new algorithms. *IEEE Transactions on Computational Imaging* 4, 3 (2018), 366–381.

M Geier, A Greiner, and JG Korvink. 2006. Cascaded digital lattice Boltzmann automata for high Reynolds number flow. *Physical Review. E* 73, 6.2 (2006), 066705–066705.

Abhinav Golas, Rahul Narain, Jason Sewall, Pavel Krajcevski, Pradeep Dubey, and Ming Lin. 2012. Large-scale Fluid Simulation Using Velocity-vorticity Domain Decomposition. *ACM Transactions on Graphics* 31, 6, Article 148 (Nov. 2012), 9 pages.

Karol Gregor and Yann LeCun. 2010. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*. 399–406.

Xiaoxiao Guo, Wei Li, and Francesco Iorio. 2016. Convolutional neural networks for steady flow approximation. In *CM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 481–490.

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

SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, Markus Gross, et al. 2015. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics* 34, 6, Article 199 (Oct. 2015), 9 pages.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The Affine Particle-in-cell Method. *ACM Transactions on Graphics* 34, 4, Article 51 (July 2015), 10 pages.

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

Tero Karras. 2012. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Conference on High-Performance Graphics*. ACM/Eurographics, 33–37.

Ladislav Kavan, Dan Gerszewski, Adam Bargteil, and Peter-Pike Sloan. 2011. Physics-inspired Upsampling for Cloth Simulation in Games. *ACM Transactions on Graphics* 30, 4, Article 93 (July 2011), 10 pages.

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum* 38, 2 (2019).

ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. 2005. FlowFixer: Using BFECC for Fluid Simulation. In *Eurographics Conference on Natural Phenomena*. 51–56.

Doyub Kim, Oh-young Song, and Hyeong-Seok Ko. 2008a. A Semi-Lagrangian CIP Fluid Solver without Dimensional Splitting. In *Computer Graphics Forum*, Vol. 27. 467–475.

Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics* 32, 4, Article 62 (July 2013), 9 pages.

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 2008b. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics* 27, 3 (Aug. 2008), 1–6.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Bryan M Klingner, Bryan E Feldman, Nuttapong Chentanez, and James F O'brien. 2006. Fluid animation with dynamic meshes. *ACM Transactions on Graphics* 25, 3 (July 2006), 820–825.

Michael Lentine, Wen Zheng, and Ronald Fedkiw. 2010. A Novel Algorithm for Incompressible Flow Using Only a Coarse Grid Projection. *ACM Transactions on Graphics* 29, 4, Article 114 (July 2010), 9 pages.

Wei Li, Kai Bai, and Xiaopei Liu. 2019. Continuous-Scale Kinetic Fluid Simulation. *IEEE Transactions on Visualization and Computer Graphics* 25, 09 (sep 2019), 2694–2709.

Wei Li, Yixin Chen, Mathieu Desbrun, Changxi Zheng, and Xiaopei Liu. 2020. Fast and Scalable Turbulent Flow Simulation with Two-Way Coupling. *ACM Transactions on Graphics* (2020).

Xiaopei Liu, Wai-Man Pang, Jing Qin, and Chi-Wing Fu. 2014. Turbulence Simulation by Adaptive Multi-Relaxation Lattice Boltzmann Modeling. *IEEE Transactions on Visualization and Computer Graphics* 20, 02 (feb 2014), 289–302.

Frank Losasso, Frédéric Gibou, and Ron Fedkiw. 2004. Simulating water and smoke with an octree data structure. 23, 3 (Aug. 2004), 457–462.

Daniel Lycett-Brown, Kai H. Luo, Ronghou Liu, and Pengmei Lv. 2014. Binary droplet collision simulations by a multiphase cascaded lattice Boltzmann method. *Physics of Fluids* 26 (2014), 023303.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. 2009. Energy-preserving Integrators for Fluid Animation. *ACM Transactions on Graphics* 28, 3, Article 38 (July 2009), 8 pages.

CUDA Nvidia. 2008. Cublas library. *NVIDIA Corporation, Santa Clara, California* 15, 27 (2008), 31.

Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

Sang Il Park and Myoung Jun Kim. 2005. Vortex Fluid for Gaseous Phenomena. In *Symposium on Computer Animation*. 261–270.

Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. 2010. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics* 29, 4, Article 66 (July 2010), 13 pages.

Andreas Peer, Markus Ihmsen, Jens Cornelis, and Matthias Teschner. 2015. An Implicit Viscosity Formulation for SPH Fluids. *ACM Transactions on Graphics* 34, 4, Article 114 (July 2015), 10 pages.

Tobias Pfaff, Nils Thuerey, Jonathan Cohen, Sarah Tariq, and Markus Gross. 2010. Scalable Fluid Simulation Using Anisotropic Turbulence Particles. *ACM Transactions on Graphics* 29, 6, Article 174 (Dec. 2010), 8 pages.

Tobias Pfaff, Nils Thuerey, and Markus Gross. 2012. Lagrangian Vortex Sheets for Animating Fluids. *ACM Transactions on Graphics* 31, 4, Article 112 (July 2012), 8 pages.

Ziyin Qu, Xinxin Zhang, Ming Gao, Chenfanfu Jiang, and Baoquan Chen. 2019. Efficient and conservative fluids using bidirectional mapping. *ACM Transactions on Graphics (TOG)* 38, 4, Article 128 (July 2019), 12 pages.

Karthik Raveendran, Chris Wojtan, and Greg Turk. 2011. Hybrid Smoothed Particle Hydrodynamics. In *Symposium on Computer Animation*. 33–42.

Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. 2018. Example-based turbulence style transfer. *ACM Transactions on Graphics* 37, 4, Article 84 (July 2018), 9 pages.

H. Schechter and R. Bridson. 2008. Evolving Sub-grid Turbulence for Smoke Animation. In *Symposium on Computer Animation*. 1–7.

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An Unconditionally Stable MacCormack Method. *Journal of Scientific Computing*

35, 2-3 (2008), 350–371.

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A Sparse Paged Grid Structure Applied to Adaptive Smoke Simulation. *ACM Transactions on Graphics* 33, 6, Article 205 (Nov. 2014), 12 pages.

B. Solenthaler and R. Pajarola. 2009. Predictive-corrective Incompressible SPH. *ACM Transactions on Graphics* 28, 3, Article 40, 6 pages.

Jos Stam. 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 121–128.

Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace condensation: full space adaptivity for subspace deformations. *ACM Transactions on Graphics* 34, 4, Article 76 (July 2015), 9 pages.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating Eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, Vol. 70. 3424–3433.

Kiwon Um, Xiangyu Hu, and Nils Thuerey. 2018. Liquid splash modeling with neural networks. In *Computer Graphics Forum*, Vol. 37. 171–182.

Nobuyuki Umetani and Bernd Bickel. 2018. Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics* 37, 4, Article 89 (July 2018), 10 pages.

Christoph Von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2013. An efficient construction of reduced deformable objects. *ACM Transactions on Graphics* 32, 6, Article 213 (Nov. 2013), 10 pages.

Steffen Weißmann and Ulrich Pinkall. 2010a. Filament-based Smoke with Vortex Shedding and Variational Reconnection. *ACM Transactions on Graphics* 29, 4, Article 115 (July 2010), 12 pages.

Steffen Weißmann and Ulrich Pinkall. 2010b. Filament-based Smoke with Vortex Shedding and Variational Reconnection. *ACM Transactions on Graphics* 29, 4, Article 115 (July 2010), 12 pages.

Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. 2019. A Multi-Pass GAN for Fluid Flow Super-Resolution. *arXiv preprint arXiv:1906.01689* (2019).

Rene Winchenbach, Hendrik Hochstetter, and Andreas Kolb. 2017. Infinite Continuous Adaptivity for Incompressible SPH. *ACM Transactions on Graphics* 36, 4, Article 102 (July 2017), 10 pages.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. *ACM Transactions on Graphics* 37, 4, Article 95 (July 2018), 15 pages.

Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. 2010. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing* 19, 11 (2010), 2861–2873.

Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski. 2018. An Advection-reflection Solver for Detail-preserving Fluid Simulation. *ACM Transactions on Graphics* 37, 4, Article 85 (July 2018), 8 pages.

Xinxin Zhang and Robert Bridson. 2014. A PPPM Fast Summation Method for Fluids and Beyond. *ACM Transactions on Graphics* 33, 6, Article 206 (Nov. 2014), 11 pages.

Xinxin Zhang, Robert Bridson, and Chen Greif. 2015. Restoring the Missing Vorticity in Advection-projection Fluid Solvers. *ACM Transactions on Graphics* 34, 4, Article 52 (July 2015), 8 pages.

Xinxin Zhang, Minchen Li, and Robert Bridson. 2016. Resolving Fluid Boundary Layers with Particle Strength Exchange and Weak Adaptivity. *ACM Transactions on Graphics* 35, 4, Article 76 (July 2016), 8 pages.

Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. 2013. A New Grid Structure for Domain Extension. *ACM Transactions on Graphics* 32, 4, Article 63 (July 2013), 12 pages.

Yongning Zhu and Robert Bridson. 2005. Animating Sand As a Fluid. *ACM Transactions on Graphics* 24, 3, 965–972.