The Visual Computer (2020) 36:1385–1399
https://doi.org/10.1007/s00371-019-01738-y

ORIGINAL ARTICLE

Deep generative smoke simulator: connecting simulated and real data

Jinghuan Wen[1] · **Huimin Ma**[1] · **Xiong Luo**[2]

Published online: 29 August 2019

Abstract
We propose a novel generative adversarial architecture to generate realistic smoke sequences. Physically based smoke simulation methods are difficult to match with real-captured data since smoke is vulnerable to disturbance. In our work, we design a generator that takes into account the temporal movement of smoke as well as detailed structures. With the help of convolutional neural networks and long short-term memory-based autoencoder, our generator can predict the future frames using temporal information while preserving details. We use generative adversarial networks to train the model on both simulated and real-captured data and propose a combined loss function that reflects both the physical laws and the data distributions. We also demonstrate a multi-phase training strategy that significantly speeds up convergence and increases stability of training on real-captured data. To test our approach, we set up experiments to capture real smoke sequences and show that our method can achieve realistic visual effects.

Keywords Smoke simulation · Generative adversarial networks · Real data · Autoencoder · LSTM

1 Introduction

Generating realistic and detailed smoke is a long-standing challenge for smoke simulation in computer graphics. Most physically based smoke simulation methods improve accuracy by using finer meshes, or using detail preserving methods such as vorticity confinement [12]. Physically based methods suffer from high computational costs. And it is difficult for these methods to synthesize a smoke sequence similar to given real-captured one, because smoke is susceptible to disturbance. Moreover, it is difficult to determine the exact physical parameters in real scenes.

Recently, data-driven methods are showing advantages in producing more detailed and diverse results. Data-driven methods extend fluid simulation in three ways: accelerating simulation steps that require massive computation [19,55], synthesizing high-resolution details based on low-resolution simulation [9,60], and predicting future states of input initial states [11,30,59]. Although real-captured data are not used in these methods, it is possible to train data-driven models over real data and produce more realistic results. However, a new architecture is needed to combine both simulated data and real-captured data into training process. Also, such architecture should be able to generate sequences that are similar to real data while being aware of physical laws.

In this paper, we present a novel network architecture that generates realistic sequences, as shown in Fig. 1. The network is trained on both simulated data and real-captured video data. Our generator takes into account short-term and long-term transformations of overall smoke shape, in addition to a detail preservation network that increases the detailed movement of smoke. We use generative adversarial networks (GAN) and a combined loss that consists of physical loss and adversarial loss to make the simulation physically accurate and similar to real-captured data.

The purpose of our method is to predict future sequence of a given previous sequence. Our generator takes an initial sequence as input and generates subsequent sequences, which is similar to a video prediction problem. There exist networks based on long short-term memory (LSTM) autoencoder for natural video prediction [31,51]. However, directly

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00371-019-01738-y) contains supplementary material, which is available to authorized users.

✉ Huimin Ma
mhmpub@tsinghua.edu.cn

Jinghuan Wen
wenjh14@mails.tsinghua.edu.cn

Xiong Luo
xluo@ustb.edu.cn

[1] Tsinghua University, Beijing, China

[2] University of Science and Technology Beijing, Beijing, China

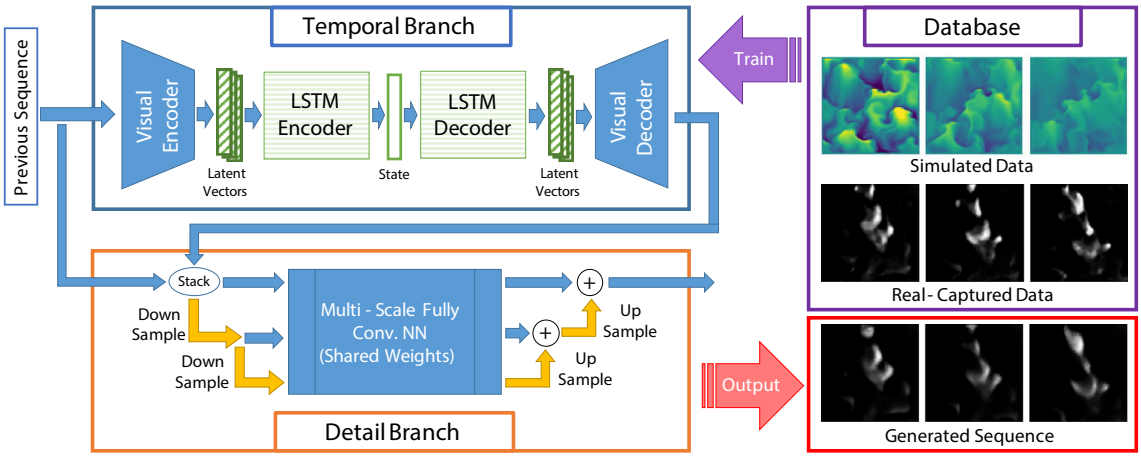


Fig. 1 General architecture of the generator proposed in this paper. The generator is composed of a temporal branch and a detail branch. Our network is trained on a database consisting of both simulated and real data and can produce smoke image sequences that are similar to real data

applying such methods to smoke simulation will generate blurred and physically incorrect results. We propose a generator structure consisting of two branches: a temporal branch and a detail branch. The temporal branch uses convolutional neural networks (CNN) as visual autoencoder and LSTM networks as sequence autoencoder. The combination of visual autoencoder and sequence autoencoder produces a general estimation of the sequence and meanwhile limits the size of state variables in LSTM for less memory usage. The detail branch is a multi-scale fully convolutional network, which takes into account the physical invariance of smoke transformation on different scales.

Generative adversarial networks show promising results on image generation [42] and video prediction [28,34]. These methods train a generator network and a discriminator network alternately, with generator trying to generate fake data similar to ground truth and the discriminator trying to distinguish between fake data and the ground truth. We use GAN to enhance the reality of generated sequences by sending discriminator real-captured data as well as simulated data.

Another challenge of our smoke simulation network is the difficulty judging whether a generated sequence is realistic. We hope the generated sequences to have similar characteristics to real-captured data, as well as being physically correct. We use an adversarial loss term to force the output similar to real-captured data and an error term to limit the output to physical laws. We also use L_1 loss and gradient difference loss to reduce blurred results, as introduced in [34].

To summarize, the major contributions of this paper are:

- a smoke simulation method connecting real-captured smoke data and physically based simulation data, based on deep generative adversarial networks,
- a novel generative smoke simulation network consisting of a temporal branch based on visual autoencoder and LSTM autoencoder, and a detail branch composed of multi-scale fully convolutional layers,
- and a combined loss for adversarial training of fluid simulation that reflects both the physical laws and the data distribution.

2 Related work

Physically based smoke simulation methods typically solve discretized Navier–Stokes (N–S) equations over a regular or irregular grid. Stam's work [52] introduced a stable fluid simulation method. It is later extended by Fedkiw [12], which gives a basic routine of smoke simulation, consisting of a semi-Lagrangian advection step, a diffusion step, a pressure projection step, and a vorticity confinement step. The major challenge of physically based smoke simulation, especially Euler methods, is that details such as turbulence vanish along simulation. To preserve details, there are many researches that focus on improving accuracy of advection [21,29,47], or restoring missing vorticity by solving vorticity–velocity form of Navier–Stocks equations [58,62]. There are vortex particle [46,61], vortex filament [2,3,36], and vortex sheets methods [7,41] that use the Lagrangian nature of vorticity and other methods that are widely used to improve the accuracy and reality of smoke simulation [6,26,63]. Physically based simulation methods also suffer from high computational costs. Many works focus on this issue, such as developing fast solvers [35] and using super-resolution techniques to generate higher-resolution results based on lower ones [24,37]. In this paper, we use physically based simulation to generate a massive amount of training data to train an initial smoke generator, which is further extended by fine-tuning on real data.

Data-driven fluid simulation extends the diversity and randomness of fluid synthesis. Instead of solving fixed partial differential equations (PDEs) iteratively, data-driven methods train models over large dataset. These models extract the characteristic of fluid flows and make it possible to synthesize fluid flow using real-captured data. This is especially important for smoke simulations where smoke simulations are more diverse and random than liquids. Data-driven methods can accelerate computational heavy steps in fluid simulations, e.g., regression forests are introduced to predict the SPH force of fluid simulation [19], and CNNs are introduced to solve pressure projections [55]. Um et al. [56] use neural networks to model liquid splash. There are also data-driven super-resolution methods that refine a coarse simulated sequence to a higher resolution [9,60]. However, these methods require an underlying low-resolution simulation, which makes it difficult for generating flow sequences that match real-captured ones. Another method by Wiewel et al. [59] is proposed to predict the future fluid flow sequences based on CNN and LSTM, which is similar to the temporal branch of our method. Kim et al. [22] present a generative network for fluid simulation using CNN. The novelty of our method includes introducing a fully convolutional detail preserving network that takes into account the physical invariance, a combined loss that makes our model more realistic as well as being aware of physical laws, and an adversarial training process that helps our model extract characteristics of real data.

Connecting physically based methods to real-captured data is quite a challenge. The exact physical parameters of certain specified scene are often difficult to determine and may not be constant in space and time. In addition, smoke is vulnerable to disturbance. A very small disturbance may seriously affect the movement of smoke over a long period of simulation. Previous works [14,40] show how to recover the 3D density field of fluids. Wang et al. [57] couple real video data with SPH system to produce guided simulation of fluids. A research by Gregson et al. [15] shows an interesting way to connect forward and inverse problems in fluids, by applying a velocity estimation method to track the movement of fluids. This method is used in our work to process the real-captured data.

Another topic in using real-captured data is to make generated images having similar characteristics to real data, rather than only comparing them pixel by pixel. There are researches that look up similar sequences or patches in databases [9,39]. Interpolation between key frames can also synthesize sequence from given data [8,53]. In our work, we use adversarial training in the final phase. Therefore, our model is not forced to match the given sequence exactly, but generates the future sequences based on statistical features of the training dataset.

Generative Adversarial Networks have shown promising results in image and video generation. The original GAN is introduced by Goodfellow et al. in [13]. Later on, a deep convolutional GAN (DCGAN) is introduced as a basic generator structure [42]. However, these methods aim at generating images or videos out of a random noise, which cannot be directly applied on our prediction problem. Conditional GAN (cGAN) is developed to generate images under certain conditions [38] and has many applications on image synthesis such as [18]. There are also temporal GANs that generate predicted future sequences [34,45,59]. Although adversarial networks show interesting visual results, training the original form of GAN is unstable and requires careful adjustment on parameters. The major issue as stated in [4] is that the original GAN training minimizes the Jensen–Shannon (JS) divergence between generated data and real data distributions, whose gradient tends to be zero when two distributions are barely overlapped. A Wasserstein GAN is introduced to minimize Earth Mover's (EM) distance (or Wasserstein distance), which is proved to have better stability. Further, an improved WGAN is introduced to solve the lack of divergence of weights in the network by using gradient penalty instead of weight clipping, which is referred to as WGAN-GP [16]. For better stability and easier training, the adversarial networks in this paper are based on WGAN-GP, although our method can fluently transport to other GAN architectures.

Video prediction is another related topic to our work. The generator proposed in this paper can be seen as a video predictor, although our generator takes into account physical invariance and physical laws. First work on natural video prediction is proposed using a recurrent convolutional neural network (rCNN) [43]. Mathieu et al. introduced a deep generative video prediction method in [34] that uses a multi-scale structure. In this paper, we design a multi-scale detail preserving network sharing the similar idea, in addition to the consideration of physical invariance. [34] also proposed an L_1-norm and a gradient-based loss term for less blurred effect in prediction problems. Srivastava et al. introduced an LSTM autoencoder for video representations [51]. There are also similar structures for video prediction such as CNN-LSTM-deCNN [31] and video pixel networks [20]. The LSTM autoencoder, together with a CNN based visual autoencoder, is adopted in the temporal branch of our generator.

3 Temporal and detail generator networks

The generator of our model consists of a temporal branch based on visual autoencoder and LSTM autoencoder and a detail branch composed of multi-scale fully convolution layers. The temporal branch generates a rough sequence which is further refined by the detail network.

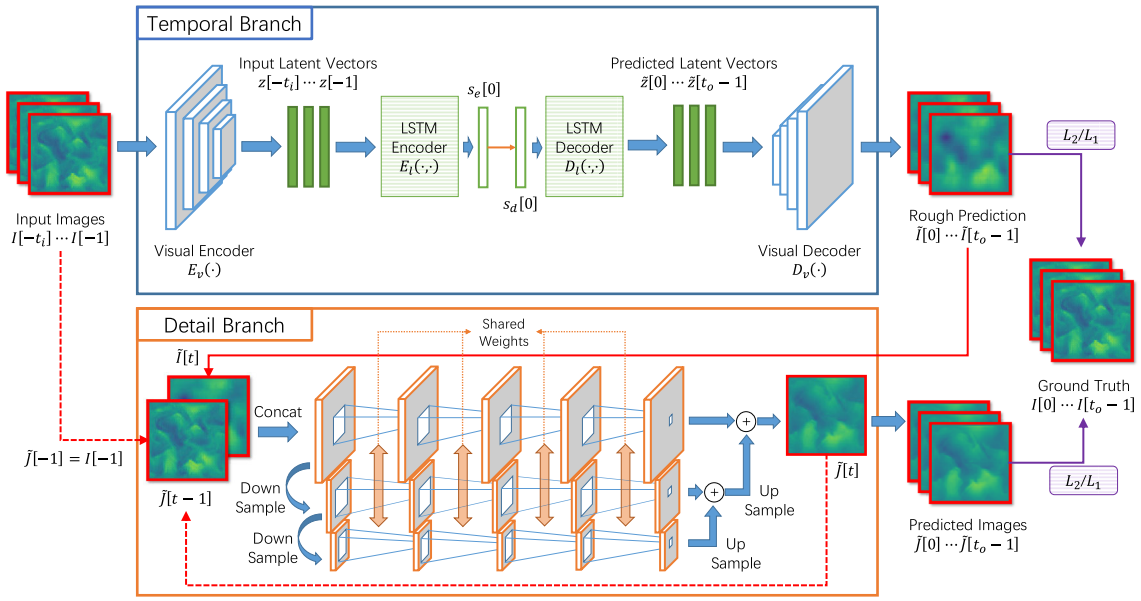


Fig. 2 Proposed generator. The generator consists of two branches. The temporal branch (above) combines visual autoencoder and LSTM autoencoder and predicts a rough image sequence $\tilde{I}[t]$. The detail branch (below) is composed of multi-scale fully convolutional networks. Convolutional kernel weights are shared on different scales. The detail branch runs iteratively on each frame, refining $\tilde{I}[t]$ to $\tilde{J}[t]$, and output predicted sequence $\tilde{J}[t]$. All of the images $I[t]$, $\tilde{I}[t]$, $\tilde{J}[t]$ have three channels: a density channel ρ and two velocity channels u_x and u_y. The illustration above only shows the density channels

The input of the generator is a provided sequence (with length t_i) of smoke images, $I[-t_i], \ldots I[-1]$. The images here (same below) consist of three channels: a density channel ρ and two velocity channels u_x and u_y. The output of the generator is a subsequent sequence (with length t_o) of smoke images, $\tilde{J}[0], \ldots, \tilde{J}[t_o-1]$. The ground truth of output sequence is denoted by $I[0], \ldots, I[t_o-1]$. The images I and $\tilde{J}$ can be either 2D or 3D (i.e., volumetric data), and for 3D image I should consist of four channels, i.e., a density channel and three velocity channels. In this paper, we focus on 2D images due to implementation limitations, although our method can be extended to 3D images.

The structure of generator is shown in Fig. 2.

3.1 Temporal branch

The temporal branch of the generator is a combination of visual autoencoder and LSTM autoencoder, as shown in Fig. 3. Visual autoencoder is widely used in computer vision as an unsupervised learning technique to efficiently encode images. We apply the encoder on each frame $I[t]$ to get latent vector

$$z[t] = E_v(I[t]), \tag{1}$$

where E_v stands for visual encoder. And we can decode the latent vector back to the original frame

$$\hat{I}[t] = D_v(z[t]), \tag{2}$$

where D_v stands for visual decoder. An ideal autoencoder will have

$$\hat{I}[t] = I[t]. \tag{3}$$

The visual autoencoder extracts low-rank representations (i.e., the latent vectors) of images, which are then used in LSTM autoencoder to predict the future sequences. The LSTM autoencoder takes a sequence of latent vectors as input, i.e., $z[-t_i], \ldots, z[-1]$, and predicts the subsequent latent vector sequence, i.e., $\tilde{z}[0], \ldots, \tilde{z}[t_o-1]$. LSTM autoencoder also contains an encoder and a decoder. The LSTM encoder extracts the representation of the final state of the input sequence, and such state (denoted by $s_e[0]$) is copied to LSTM decoder as initial state. The LSTM decoder runs iteratively and predicts a frame $\tilde{z}[t]$ at each step, and the predicted latent vectors can then be decoded to images, i.e.,

$$\tilde{I}[t] = D_v(\tilde{z}[t]). \tag{4}$$

The underlying structure of visual autoencoder is CNN [27]. The encoder consists of several convolutional layers and pooling layers, while the decoder consists of several transposed convolutional layers. Activation functions such as ReLU are applied following each of the convolutional layers. There are some pre-trained convolutional units such as VGG

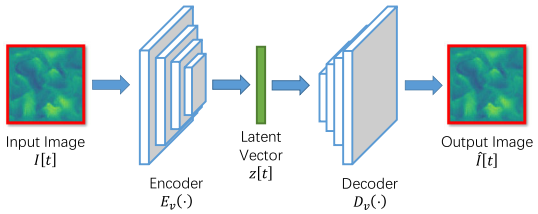


Fig. 3 Visual autoencoder. The encoder E_v encodes the input image $I[t]$ to a latent vector $z[t]$, while the decoder D_v restores image $\hat{I}[t]$ from latent vector $z[t]$

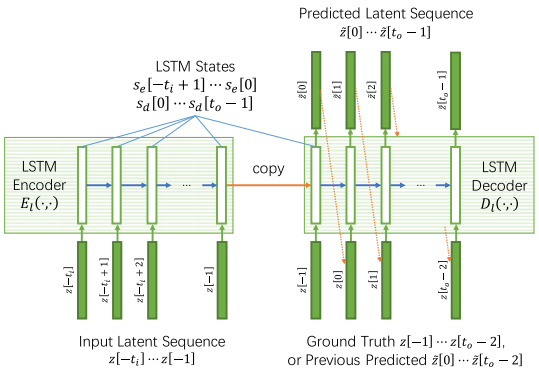


Fig. 4 LSTM autoencoder for unsupervised video representation. The encoder extracts the final state $s_e[0]$ of input sequence $z[t]$ for $t < 0$. The final state is then copied to the decoder as initial state $s_d[0]$. Decoder runs iteratively and generates predicted future sequence $z[t]$ for $t \geq 0$

[50]; however, we find a simple CNN with much less depth (4 convolutional layers in our implementation) works well in our method. There are other autoencoders that work well on various problems, such as variational autoencoder (VAE) [25]. However, note that the purpose of using autoencoder in the temporal branch is to reduce the dimension of input image, and we hope to reconstruct the original image from the extracted latent vector. Therefore, random autoencoders such as VAE should not be adopted. It is also demonstrated in [59] that VAE is not suitable for temporal problems.

Since our model is composed of a complicated set of networks, we designed several training phases for better convergence, which will be explained in Sect. 4. Here, as the first training phase, we pre-train the visual autoencoder before training the rest parts of generator using random generated, independent smoke images (instead of sequences). We define a loss function that considers each smoke image independently:

$$\begin{aligned} L_{E_v,D_v} &= \mathbb{E}_{I\sim p_{\text{sim}}}\left[\|\hat{I} - I\|\right] \\ &= \mathbb{E}_{I\sim p_{\text{sim}}}\left[\|D_v(E_v(I)) - I\|\right], \end{aligned} \tag{5}$$

where p_{sim} is the data distribution of simulated smoke images, $\mathbb{E}_{I\sim p_{\text{sim}}}[\cdot]$ is the expectation over the distribution of simulated smoke images, and $\|\cdot\|$ is L_1 or L_2 norm.

The LSTM autoencoder consists of two LSTM units. LSTM is a recurrent neural network (RNN) that is composed of a cell, an input gate, an output gate, and a forget gate [17]. LSTM is successfully used in neural networks on sequences. In our method, we use LSTM to solve the sequential data prediction problem, although other RNN models can be used as alternatives. The LSTM autoencoder structure for unsupervised video representation is shown in Fig. 4. This structure and its training strategy are first presented in [51].

The LSTM encoder runs iteratively and updates its own state, which can be represented as

$$s_e[t+1] = E_l(z[t], s_e[t]), \tag{6}$$

where $s_e[t]$ is the state of encoder after processing $z[t]$ and E_l is the LSTM encoder. In our method, the initial state $s_e[-t_i]$ can be an arbitrary state vector. For convenience, we set $s_e[-t_i] = 0$. The LSTM decoder also runs iteratively to generate output as well as updating its own state, which can be represented as

$$\langle \tilde{z}[t], s_d[t+1] \rangle = D_l(z[t-1], s_d[t]), \tag{7}$$

where $s_d[t]$ is the state of decoder before generating the output of $\tilde{z}[t]$ and D_l is the LSTM decoder. The connection between the LSTM encoder and the LSTM decoder is that the final state of the encoder is copied to the initial state of the decoder, i.e.,

$$s_d[0] = s_e[0]. \tag{8}$$

To train an LSTM autoencoder, we first train an encoder–decoder pair that encodes a sequence and decodes it back. An ideal encoder and the corresponding decoder $D_l^*(\cdot)$ should satisfy

$$\tilde{z}[t] = z[t], \quad t = -t_i, \ldots, -1. \tag{9}$$

Therefore, for this training phase (phase 2) we have loss function

$$\begin{aligned} L_{E_l} &= \mathbb{E}_{I\sim p_{\text{sim}}}\left[\sum_{t=-t_i}^{-1} \|\tilde{z}[t] - z[t]\|\right] \\ &= \mathbb{E}_{I\sim p_{\text{sim}}}\left[\sum_{t=-t_i}^{-1} \|D_l^*(E_l(E_v(I[t]))) - E_v(I[t])\|\right], \end{aligned} \tag{10}$$

where $E_l(\cdot)$ and $D_l^*(\cdot)$ are simplified form of $E_l(\cdot,\cdot)$ and $D_l^*(\cdot,\cdot)$ that omit the states, and $z[t]$ is generated from pre-trained visual autoencoder E_v. This training phase is for the encoder to learn a valid representation of sequences. The

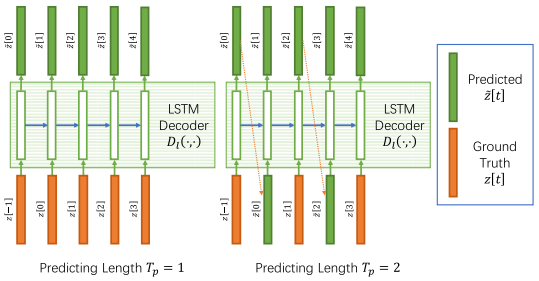


Fig. 5 Dynamic predicting length for training LSTM decoder and detail branch. *Left:* At first, when $T_p = 1$, the ground truth is provided on each iteration. *Right:* As training step grows, a larger T_p is used. And the ground truth is provided every T_p iteration

temporary decoder D_l^* is discarded, and a predictive decoder is trained using future sequences, i.e.,

$$\begin{aligned} s_e[t+1] &= E_l(\tilde{z}[t], s_e[t]), \quad t = -t_i, \ldots, -1 \\ s_d[0] &= s_e[0] \\ \langle \tilde{z}[t], s_d[t-1] \rangle &= D_l(z[t-1], s_d[t]), \quad t = 0, \ldots, t_o - 1\,, \end{aligned} \tag{11}$$

where $D_l(\cdot,\cdot)$ is the decoder that can predict future sequences. We hope the predicted sequence is close to ground truth, which gives a loss function for phase 3:

$$\begin{aligned} L_{D_l} &= \mathbb{E}_{I\sim p_{\text{sim}}}\left[\sum_{t=0}^{t_o-1} \|\tilde{z}[t] - z[t]\|\right] \\ &= \mathbb{E}_{I\sim p_{\text{sim}}}\left[\sum_{t=0}^{t_o-1} \|D_l(E_l(E_v(I[t]))) - E_v(I[t])\|\right]. \end{aligned} \tag{12}$$

It should be noticed that the ground truth $z[t], t = 0, \ldots, t_o - 1$ should be provided during training. However, there is no ground truth during the test (nor in actual use). Therefore, an iterative prediction function substitutes $z[t]$ by $\tilde{z}[t]$ in Eq. 11, giving

$$\langle \tilde{z}[t], s_d[t-1] \rangle = D_l(\tilde{z}[t-1], s_d[t]), \quad t = 0, \ldots, t_o - 1\,. \tag{13}$$

In practice, however, we find that splitting training and test is not a good strategy, because the model tends to learn only short-term transformation of sequence if ground truth is provided for every iteration of D_l. We apply a dynamic predicting length that gradually increases during training process, as shown in Fig. 5. A predicting length of T_p means we use iterative method (Eq. 13) in training for continuous T_p frames (i.e., the 0-th to $(T_p - 1)$-th frames) and reset the input to ground truth for the T_p-th frame. Therefore, as T_p grows, the model is forced to learn longer sequences. When $T_p = t_o$, the decoder decodes as it does during the test. The same strategy is also used in the training of detail branch. To summarize, the algorithm of temporal branch is listed in Algorithm 1.

Algorithm 1: Temporal Branch

Input: Input sequence: $I[t] \mid t = -t_i, \ldots, -1$
Output: Rough prediction: $\tilde{I}[t] \mid t = 0, \ldots, t_o - 1$

begin
  for $t = -t_i, \ldots, -1$ **do**
    $z[t] \longleftarrow E_v(I[t])$
    $s_e[t+1] \longleftarrow E_l(z[t], s_e[t])$
  $s_d[0] \longleftarrow s_e[0]$
  $\tilde{z}[-1] \longleftarrow z[-1]$
  for $t = 0, \ldots, t_o - 1$ **do**
    $\langle \tilde{z}[t], s_d[t-1] \rangle = D_l(\tilde{z}[t-1], s_d[t])$
    $\tilde{I}[t] \longleftarrow D_v(\tilde{z}[t])$

Algorithm 2: Detail Branch

Input: Rough prediction: $\tilde{I}[t], t = 0, \ldots, t_o - 1$, and $I[-1]$
Output: Refined Prediction: $\tilde{J}[t], t = 0, \ldots, t_o - 1$

begin
  $\tilde{J}[-1] \longleftarrow I[-1]$
  for $t = 0, \ldots, t_o - 1$ **do**
    $J^{(0)}[t] \longleftarrow Concatenate(\tilde{J}[t-1], \tilde{I}[t])$
    for $i = 1, \ldots, n_{scales}$ **do**
      $J^{(i)}[t] \longleftarrow DownSample(J^{(i-1)}[t])$
    for $i = 0, \ldots, n_{scales}$ **do**
      $\tilde{J}^{(i)}[t] \longleftarrow FullyConv(J^{(i)}[t])$
    $\tilde{J}[t] \longleftarrow \tilde{J}^{(0)}[t]$
    for $i = 1, \ldots, n_{scales}$ **do**
      $\tilde{J}[t] \longleftarrow \tilde{J}[t] + UpSample(J^{(i)})$

3.2 Detail branch

The output of temporal branch is an approximation of future image sequences, although it is usually blurred. The visual autoencoder reduces the dimension of input images, making the number of trainable parameters feasible in implementation. However, it prevents the network from generating detailed results. To address this issue, we add a detail branch to the generator that refines the detailed movement of smoke. The detail branch is composed of multi-scale fully convolutional networks, as shown in Fig. 2. We use fully convolutional network without pooling for better performance on detail preserving. We use multi-scale structure according to the fact that there is certain physical invariance at different scales. For example, the advection schemes are the same regardless of the scale. Therefore, we apply shared convo-

lutional weights over multiple scales. There do exist some physical quantities that vary in scales. Thus, we add four 1×1 convolution kernels before and after down-sampling and up-sampling. We also apply residual structures [23] on multiple scales, in which lower resolution data are added back to higher-resolution data after convolution, to improve the accuracy of the network.

The input of detail branch includes the rough sequence from the temporal branch $\tilde{I}[t], t = 0, \ldots, t_o$, as well as the last frame as reference. The output of detail network is the refined sequence $\tilde{J}[t], t = 0, \ldots, t_o$. The full algorithm of detail branch is listed in Algorithm 2. $J^{(i)}[t]$ and $\tilde{J}^{(i)}[t]$ are data before and after the shared convolutional layers at the i-th scale. The loss for training detail branch (in phase 4) is the difference between $\tilde{J}[t]$ and ground truth $I[t]$, which will be further discussed in Sect. 4.

4 Adversarial training on real data

4.1 Generative adversarial networks

We use adversarial training to improve the generator by making the generated smoke similar to real smoke. Adversarial networks consist of two networks: a generator G and a discriminator D. Instead of considering only the L_2 or L_1 distance to ground truth, the discriminator extracts features of smoke sequences and uses these features to determine whether a smoke sequence is real or fake. For better training, we use WGAN with gradient penalty proposed in [16], with the loss defined as

$$L_D = \mathbb{E}_{\tilde{J}\sim p_{\text{gen}}}\left[D(\tilde{J})\right] - \mathbb{E}_{I\sim p_{\text{gt}}}\left[D(I)\right] + \lambda \mathbb{E}_{\check{I}\sim p_{\text{interp}}}\left[(\|\nabla_{\check{I}} D(\check{I})\|_2 - 1)^2\right], \tag{14}$$

where $\tilde{J}$ is the generated image sequence and its distribution is p_{gen}. I is the ground truth sequence either from simulated data $p_{gt} = p_{\text{sim}}$ or real-captured data $p_{gt} = p_{\text{real}}$. $\check{I}$ is sampled from the interpolation between distributions p_{gen} and p_{gt}. λ is the penalty coefficient. We use $\lambda = 10$ as suggested in [16].

The discriminator network in our method is based on CNN. In GAN training, discriminator and generator should be balanced. Therefore, a simple discriminator with only 4 layers of 3D convolution is used. Instead of applying convolution over the entire generated sequence, we use a patch discriminator that runs on a short sequence at a time. We apply the patch discriminator on a sliding window over the sequence and generate a probability map that reflects the probability of generated images over time. The patch discriminator is similar to the local adversarial loss proposed in [49], except that the patch in [49] is spatial while ours is temporal. We calculate a weighted sum over the probability map and obtain the output of discriminator. In practice, we find the beginning of the sequence should have larger weights because the smoke tends to vanish over a long period of time, leading to much smaller average error for long-term predictions.

4.2 Loss functions

In prediction problems, the predicted images tend to be blurred. The loss function has a significant effect on this issue. Since there are multiple possibilities of future sequences, a model trained on L_2 loss tends to fit the mean value of target distribution and median value for model trained on L_1 loss. [34] shows that L_1 loss works better in video prediction problems and introduces a new image gradient difference loss (GDL) to improve the sharpness of predicted images. The GDL between two images X and Y is defined as the difference between the gradients of the two images, i.e.,

$$L_{gdl}(X, Y) = \sum_{i,j} \Big| \big|X_{i,j} - X_{i-1,j}\big| - \big|Y_{i,j} - Y_{i-1,j}\big| \Big|^p + \sum_{i,j} \Big| \big|X_{i,j} - X_{i,j-1}\big| - \big|Y_{i,j} - Y_{i,j-1}\big| \Big|^p, \tag{15}$$

where $p = 1$ or 2 for L_1 or L_2 norm. An ordinary L_p loss is also applied, i.e.,

$$L_p(X, Y) = \|X - Y\|_p . \tag{16}$$

In our generator, the temporal branch controls the general movement of smoke, which is significant for generating proper long-term results. However, the output of temporal branch $\tilde{I}$ does not directly connect to loss computing, which causes the temporal branch to lose its function of predicting a rough sequence. We add an intermediate L_p loss term

$$L_{tp} = L_p(\tilde{I}, I). \tag{17}$$

that limits the output of temporal branch. We also apply a divergence loss term to ensure the generated sequence fits the conservation of mass, i.e.,

$$L_{\text{div}}(\mathbf{u}) = \sum_{i,j} |\nabla \cdot \mathbf{u}|^2 . \tag{18}$$

where $\mathbf{u} = (u_x, u_y)^{\mathrm{T}}$ is the velocity field. According to Gauss's law, a velocity field should have $\nabla \cdot \mathbf{u} = 0$ for mass conservation. Therefore, a large L_{div} indicates there is significant loss (or gain) of mass.

The final loss for generator is

$$L_G = -\mathbb{E}_{\tilde{J}\sim p_{\text{gen}}}\left[D(\tilde{J})\right] + L_p(\tilde{J}, I) + L_p(\tilde{I}, I) + L_{\text{div}}(\mathbf{u})\,. \tag{19}$$

4.3 Training phases

Our network includes two autoencoders (visual and LSTM autoencoders): an iterative convolutional network (detail branch) and a discriminator network. Such complicated networks should be trained under multiple phases to prevent the loss from being too large. Our major concern is that the iterative networks, including the LSTM autoencoder and the detail branch, are hard to perform end-to-end training. We address this issue by applying dynamic predicting length (shown in Fig. 5). We also find that the pre-trained autoencoders allow training to converge faster than direct end-to-end training. Therefore, we introduce a training strategy consisting of 6 phases:

- *Phase 1* Train visual autoencoder E_v and D_v, minimizing L_{E_v,D_v} (Eq. 5).
- *Phase 2* Train the LSTM encoder E_l, minimizing L_{E_l} (Eq. 10), with fixed E_v and D_v. After phase 2, discard the temporary decoder D_l^*.
- *Phase 3* Train the LSTM decoder D_l, minimizing L_{D_l} (Eq. 12), with fixed E_v, D_v, and E_l.
- *Phase 4* Train the detail branch, minimizing $L_p(\tilde{J}, I)$, with fixed E_v, D_v, E_l, and D_l. The pre-training of generator G ends in this phase.
- *Phase 5* Train G and D in conjunction. Minimize L_G (Eq. 19) for G and L_D (Eq. 14) for D. All of the training in phase 1 to 5 use simulated data.
- *Phase 6* Fine-tune the model G and D with real-captured data.

5 Implementations

5.1 Network configurations

The configurations of networks are listed in Table 1. The layers listed in the table are all convolutional layers, except that the visual decoder uses transposed convolutional layers. All of the convolutional layers use ReLU activation functions except for output layers. The size of input and output images is 64×64. More complicated networks can be used for better performance.

Our network is implemented using TensorFlow [1], and the training process runs on NVIDIA TITAN GPUs. We also implemented a Navier–Stokes smoke solver using CUDA. The NS Solver is used to generate massive amount of simulated training data. The simulated data is used for pre-training models from phases 1 to 5. Each training phase uses 800,000 sequences of simulated data. The simulated data are initialized with a random noise map, so that almost every part of the image is valuable for training. In contrast, if we use a specific scene for smoke (e.g., rising smoke scene used in many physically based simulation), the majority part of the data would be blank and would cause unnecessary computation. Finally, we fine-tune the model on a real-captured dataset of 2000 sequences.

Table 1 Network configurations

	Kernel sizes	Num. of features
Visual encoder E_v	5, 3, 3, 3	3, 8, 16, 32, 32
Visual decoder D_v	3, 3, 3, 5	32, 32, 16, 8, 3
Fully conv.	3, 5, 5, 3	6, 16, 64, 16, 3
Discriminator D	7, 5, 5, 3	3, 16, 32, 16, 1

Fig. 6 Experiment setup for capturing smoke videos. A white LED panel is used as background. A transparent box with size of 32 cm × 32 cm × 0.5 cm is used to limit the smoke moving mainly in 2D space. The special smoke generators (bottom left) can generate smoke (denser than the air) while burning

5.2 Capturing real smoke

Our adversarial networks use real-captured training data to improve their fidelity. To this end, we build a 2D real smoke dataset using the similar technique to [14,15]. A photograph of the experiment setup is shown in Fig. 6. We set up a high frame rate camera to capture the movement of real smoke. A container with size of 32 cm × 32 cm × 0.5 cm is used

Table 2 Computational time

Grid size	CFD solver (ms)	Ours (ms)
64×64	9.564	8.260

to limit the smoke moving mainly in 2D space. We use a special kind of smoke that is denser than air for better control. Thus, our smoke moves downwards instead of moving upwards like most smoke. A white LED panel is placed as a background for illumination. We assume an image formation model where light is absorbed according to the density of smoke. Therefore, the luminance of a grayscale video represents the density of smoke (in negative correlation). However, the mapping function between smoke density and image luminance is unknown; we use a simple linear model to get the density of smoke.

The captured video is undistorted and clipped, removing the distortion and redundant surroundings. We then adjust the luminance of video for better contrast between smoke and the background. We apply V-BM4D video denoising [32, 33] on the video and obtain a relatively clean video. It is resized and sampled before used as the density field of smoke. We use Horn–Schunck method [5] to get an optical flow of the video and use it as a initial guess of velocity field. We solve an optimization problem introduced in [15] to obtain the velocity field of smoke.

6 Results and discussions

6.1 Quantitative evaluation

The computational time is shown in Table 2. Both methods are implemented on GPU. Our method can accelerate simulation speed by approximately 16%.

To evaluate the quality of generated smoke sequences, we measure the L_2 and L_1 error between the predicted sequences and the ground truth, represented by mean squared error (MSE) and mean absolute error (MAE), respectively. We also measure the sharpness difference using gradient difference loss (GDL) in Eq. 15 with $p = 1$. Another metric, divergence loss in Eq. 18, is introduced to evaluate whether the models conform to the conservation of mass. The numerical results of simulated dataset are listed in Table 3. The minimal of each column is marked bold.

Table 3 is divided into 3 groups: video prediction network [34], our model pre-trained in phase 3 and phase 4, and our model trained with different loss functions. It shows that the average prediction error decreases as the network structure and the loss function become more complex. We can see a significant performance gain from phases 4 to phase 5, which indicates the effect of introducing the adversarial training.

Table 3 Average prediction error on simulated dataset

Models	MSE	MAE	GDL	L_{div}
[34] w/o Adv.	151.36	574.18	544.93	7.444
[34] w/ Adv.	83.08	429.54	**506.05**	8.078
Temporal (phase 3)	89.19	426.40	601.04	**3.263**
+Detail (phase 4)	91.53	436.65	581.80	4.475
+Adv. (phase 5)				
L_2	82.54	416.29	574.84	5.441
$L_2 + L_{\text{gdl}}$	85.80	422.77	540.38	5.497
$L_1 + L_{\text{gdl}}$	74.22	391.85	522.53	5.465
$L_1 + L_{\text{gdl}} + L_{\text{tp}}$	**74.18**	**391.74**	522.54	5.473
$L_1 + L_{\text{gdl}} + L_{\text{tp}} + L_{\text{div}}$	74.24	391.89	522.52	5.469

Loss functions also make differences in training predictive model. By training our model with L_2 or L_1 loss on simulated data, it is interesting to see an L_1 loss function minimizes the errors, even for error that based on L_2 metric (MSE). This is partly due to the fact that simulated data have sharp borders, and an L_1 loss is better at preserving sharpness. We will see in Table 4 that this is different for real data.

More experiments are done using real-captured dataset, and results are shown in Table 4. The two minimals of each columns are marked bold. We can see that the fine-tuned model has the best performance. Further, we focus on how the prediction error changes through iteration steps. To this end, we list errors of average prediction, next frame prediction, and long-term prediction in Table 4. Although video prediction network [34] performs well on next frame prediction, our model works better in long-term predictions. We should also see that our models in phase 5 have similar MSE to the model in phase 3. This is because MSE is an L_2 metric, and minimizing L_2 loss finds the mean value of target distribution. Since smoke sequences (as well as video sequences) are continuous over time, the mean value of target distribution (i.e., expectation) gives a decent estimate of future sequences. Our model in phase 3 is a temporal branch using LSTM autoencoder, and its training is based on L_2 metric. Therefore, the temporal branch plays an important role in long-term prediction.

We can also compare the error terms in Fig. 7. Our model has stable errors over time and therefore is more suitable for long-term prediction. The models without fine-tuning have similar performance on real dataset. It is due to the difference between distributions of training and test samples. The GDL and L_{div} decrease on long-term prediction. This is mostly because the smoke tends to be dissipate after 50 steps, and an image with dissipated smoke has smaller gradient and divergence.

Table 4 Prediction error on real dataset

Models	Avg. prediction error (50 frames)			Next frame prediction error			Long-term (50th frame) prediction error		
	MSE	MAE	GDL	MSE	MAE	GDL	MSE	MAE	GDL
[34] w/o Adv.	338.30	832.52	437.54	53.71	**305.20**	**190.33**	459.22	1036.26	485.14
[34] w/ Adv.	261.53	683.89	455.16	**52.82**	305.73	191.04	309.44	767.06	544.62
Temporal (phase 3)	**186.47**	576.72	**213.30**	127.78	475.55	244.53	**203.75**	615.86	**208.16**
+Detail (phase 4)	200.49	592.04	227.30	154.44	523.14	225.19	209.69	617.03	236.26
+Adv. (phase 5)									
L_2	192.80	583.31	302.76	150.60	513.04	311.77	207.70	614.85	291.56
$L_2 + L_{\text{gdl}}$	196.34	590.01	262.31	164.52	542.61	246.04	208.95	614.70	252.68
$L_1 + L_{\text{gdl}}$	187.06	569.52	252.97	113.58	430.52	223.10	204.39	615.22	247.87
$L_1 + L_{\text{gdl}} + L_{\text{tp}}$	187.13	569.11	252.30	110.90	424.47	222.52	204.95	615.02	246.38
$L_1 + L_{\text{gdl}} + L_{\text{tp}} + L_{\text{div}}$	187.28	**568.47**	251.86	111.30	425.65	223.04	204.82	**614.28**	246.23
Fine-tuned (phase 6)	**99.70**	**373.55**	**132.51**	**24.41**	**207.49**	**111.67**	**131.06**	**428.99**	**137.55**

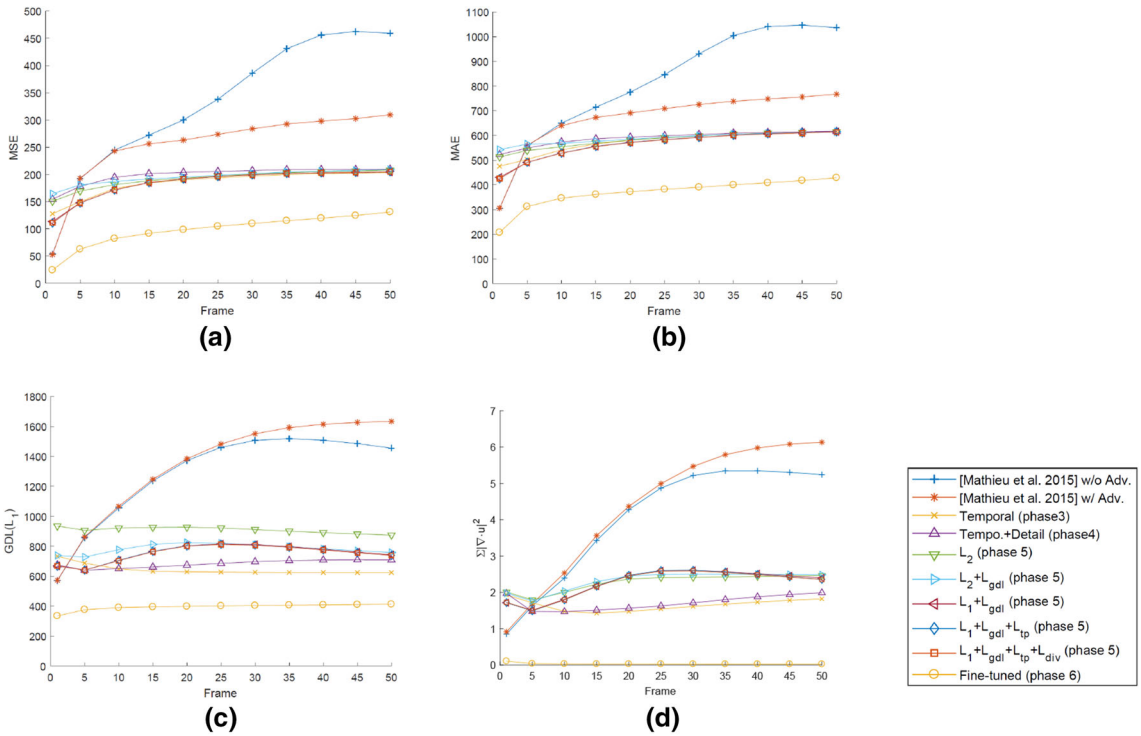


Fig. 7 Prediction errors versus time. **a** MSE. Video prediction network [34] produces very little error on the first frame, but its MSE increases rapidly after a few frames. Our models have stable MSE over time. **b** MAE. Similar to MSE. **c** GDL(L_1). The GDLs of our models hardly change with time and sometimes even decrease. **d** L_{div}. Our model preserves mass over long-term prediction

6.2 Visual results

Visual results are shown in Fig. 8. We render the output and the corresponding ground truth to show the effectiveness of our method. The generated sequence of our model shows similar shape and movement to the ground truth. There are less details generated for the frames of long-term prediction, but the overall shape remains similar. This can be seen as a compromise between short-term prediction and long-term prediction. In contrast, the video prediction network [34] shows much details on the first few frames, but fails to track the correct shape of smoke, especially for long-term prediction.

Figure 9 visualizes the density field in pseudo-color to show the effect of introducing temporal branch, detail branch, and adversarial training. The images are from the evaluation

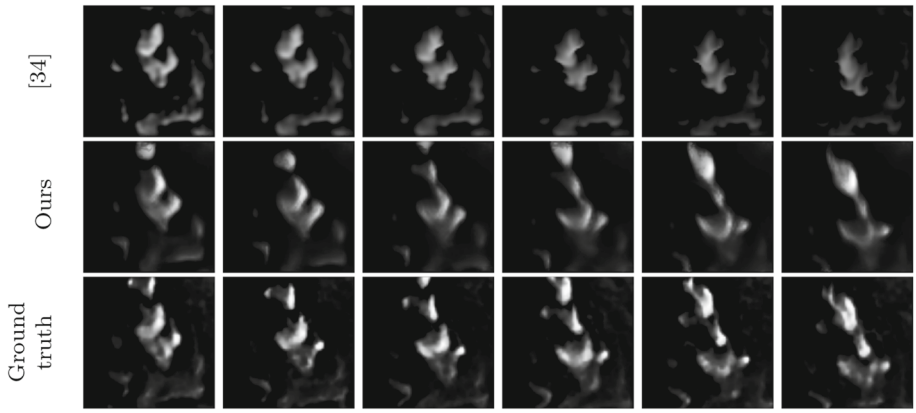


Fig. 8 Visual results of real dataset. Each row is a sequence (frames 4, 8, 12, 16, 20, 24). All images are rendered under the same renderer. Our model shows similar shape and motion to the ground truth

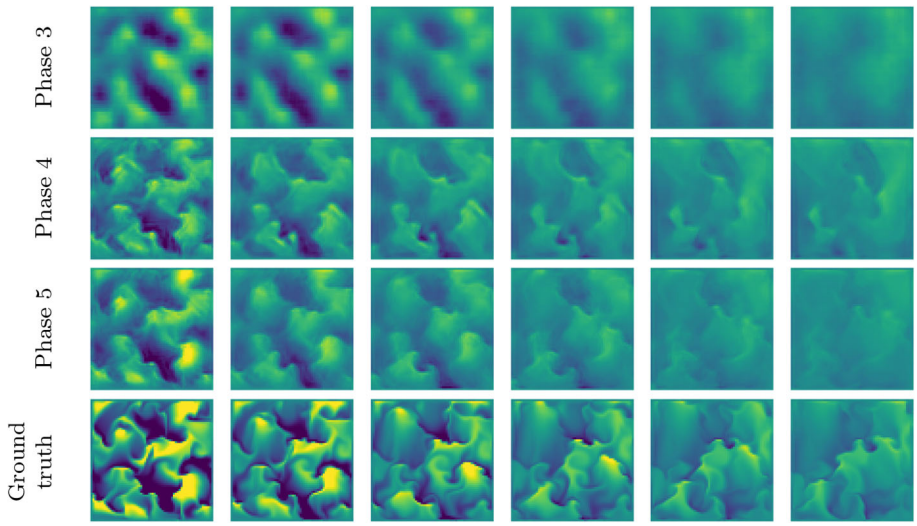


Fig. 9 Visualization of density field for results on simulated dataset. Each row is a sequence (frames 0, 5, 10, 15, 20, 25)

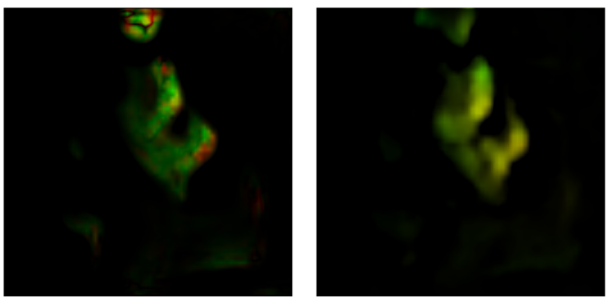


Fig. 10 Velocity field of our model and ground truth. The red and green channels of the image represent the velocity on x and y directions. Our model uses the prior knowledge to construct a detailed velocity field

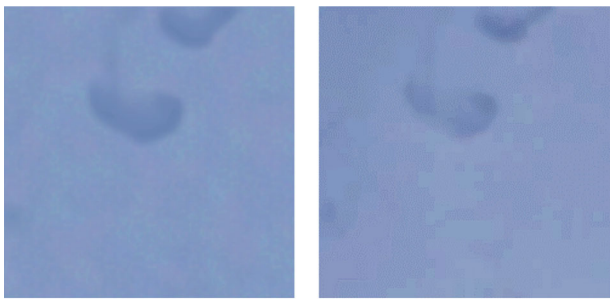


Fig. 11 A demonstration showing that our method can match real-captured data. *Left:* The next frame prediction of a sequence in real-captured test dataset. *Right:* The raw captured video frame. Both images have been adjusted for better reading using the same parameter

dataset which is simulated by N–S solver. We can see from Fig. 9 that the model consisting of only temporal branch gives blurred images. However, the overall trend of movement is the same as the real data. Phase 4 introduces the detail branch, and we can see a significant detail enhancement on each frame, even on heavily blurred image (to the right of the first row). In phase 5, a discriminator network is introduced for adversarial training. We can see that the values around peak and valley points are getting nearer to the real value. Thus, we can conclude that the temporal branch in this model is used to generate a decent estimate of future sequences. It is blurred, but on L_2 metric it is near to the real sequences. A detail branch adds details such as turbulences. Adversarial training is useful for balancing the long-term trends and short-term

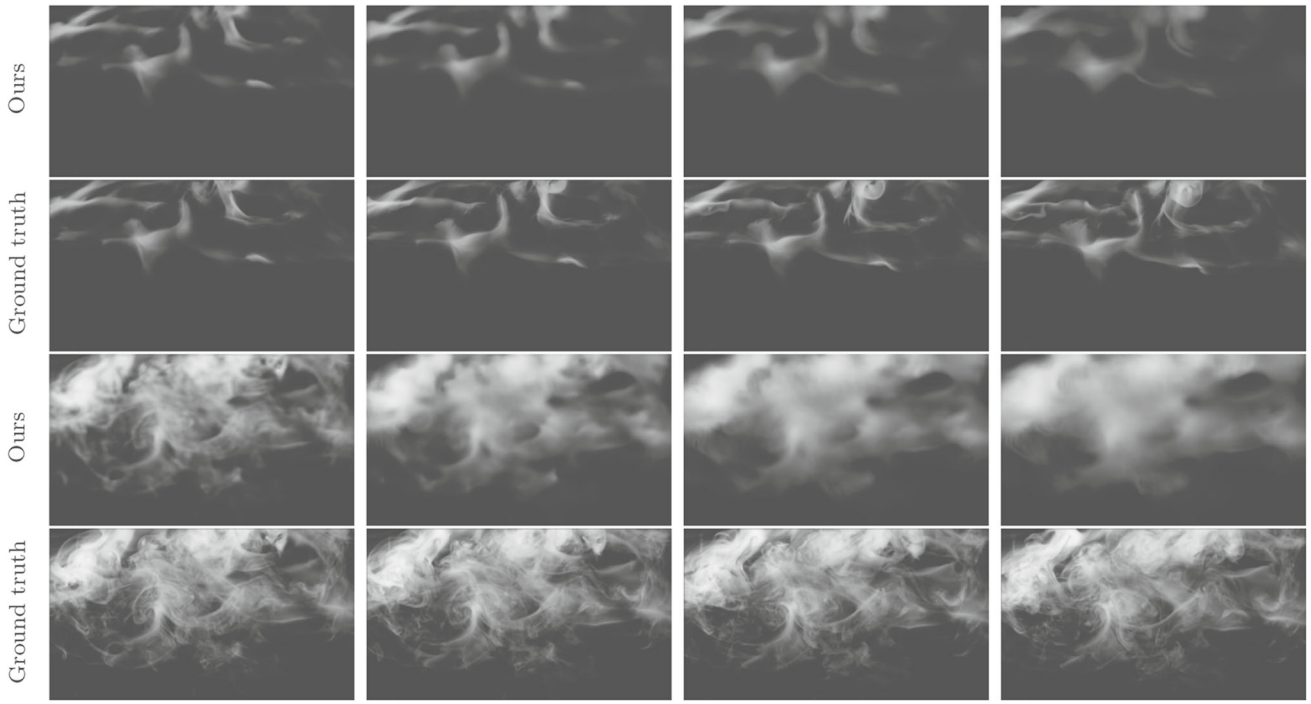


Fig. 12 Visual results on large real-captured images. For each row $t = 0, 10, 20, 30$. The images are in 1280×720

details. All of these parts are crucial for the network to learn representative features of smoke sequences.

In Fig. 10, we visualize the velocity field of our result. The velocity field of ground truth lacks details because it is calculated by solving an optimization problem. However, our model use the prior knowledge from the pre-train phases of simulated data to construct a detailed velocity field. We also render a smoke with the similar background color to the real-captured video, as shown in Fig. 11. We can see that our model can match real-captured video. A video of these visual results is found in the supplement materials.

Figure 12 shows how to use our model on larger test images. Due to the translation invariance of convolution kernel and the locality of smoke motion, we can use larger image in testing and generate arbitrarily large smoke image sequences with only forward iterations. The results in Fig. 12 are generated by replacing the LSTM model to ConvLSTM [48]. The testing data are from internet videos [10,54]. It should be noticed that the videos are smoke in 3D motion, which does not apply to the divergence loss, resulting in less realistic visual effects. Also the images tend to be more blurred with larger t.

The major limitation of this work is that we only focus on 2D smoke sequences, but a majority of smoke simulation applications require 3D scenes. Although the network structure proposed in this paper can be extended to 3D cases, the network capacity limits the target resolution and mini-batch size of training. Another issue limiting us on 2D scene is the difficulty in capturing 3D real data. There exist researches that capture real fluid data in 3D [14,15], but only for liquids and are difficult to apply on real smoke scene. Currently, using our method for 3D scene requires extra expansions, such as using 2D to 3D extension methods such as [44], or use a 3D coarse physically based solver and enhance it with our method. Also, our method is only trained on free boundary conditions to match the real-captured data. We expect future works to extend our method to 3D complicated scenes and to simulating other physical materials.

7 Conclusions

We have presented a novel architecture for generating realistic smoke sequences using deep adversarial networks. Our approach uses a temporal branch and a detail branch for learning long-term and detail representations. To ensure the practicality of our model, we set up environment to capture real smoke videos and use them to improve our model. Our model is trained using a multi-phase training strategy that can be extended to similar networks. Results show that our model is capable of learning long-term features of smoke movement as well as preserving details. Using rendering techniques, we can produce smoke sequences that match real-captured ones.

Funding This work was funded by National Key Basic Research Program of China (No. 2016YFB0100900) and National Natural Science Foundation of China (No. 61773231).

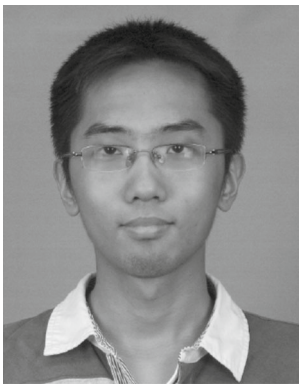
Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. OSDI **16**, 265–283 (2016)
2. Angelidis, A., Neyret, F.: Simulation of smoke based on vortex filament primitives. In: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on computer animation, ACM, pp. 87–96 (2005)
3. Angelidis, A., Neyret, F., Singh, K., Nowrouzezahrai, D.: A controllable, fast and stable basis for vortex based smoke simulation. In: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on computer animation, Eurographics Association, pp. 25–32 (2006)
4. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. (2017). arXiv:1701.07875
5. Barron, J.L., Fleet, D.J., Beauchemin, S.S.: Performance of optical flow techniques. Int. J. Comput. Vis. **12**(1), 43–77 (1994)
6. Brackbill, J., Ruppel, H.: Flip: a method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. J. Comput. Phys. **65**(2), 314–343 (1986)
7. Brochu, T., Keeler, T., Bridson, R.: Linear-time smoke animation with vortex sheet meshes. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation, Eurographics Association, pp. 87–95 (2012)
8. Browning, M., Barnes, C., Ritter, S., Finkelstein, A.: Stylized keyframe animation of fluid simulations. In: Proceedings of the workshop on non-photorealistic animation and rendering, ACM, pp. 63–70 (2014)
9. Chu, M., Thuerey, N.: Data-driven synthesis of smoke flows with cnn-based feature descriptors. ACM Trans. Graph. **36**(4), 69 (2017)
10. Digital meals Smoke atmosphere. (2012). http://digitalmeals.blogspot.com/2012/03/smoke-atmosphere.html
11. Farimani, A.B., Gomes, J., Pande, V.S.: Deep learning the physics of transport phenomena. (2017). arXiv:1709.02432
12. Fedkiw, R., Stam, J., Jensen, H.W.: Visual simulation of smoke. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques, ACM, pp. 15–22 (2001)
13. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, vol. 27, pp. 2672–2680 (2014). http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
14. Gregson, J., Krimerman, M., Hullin, M.B., Heidrich, W.: Stochastic tomography and its applications in 3d imaging of mixing fluids. ACM Trans. Graph **31**(4), 52–61 (2012)
15. Gregson, J., Ihrke, I., Thuerey, N., Heidrich, W.: From capture to simulation: connecting forward and inverse problems in fluids. ACM Trans. Graph. **33**(4), 139 (2014)
16. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Advances in Neural Information Processing Systems, vol. 30, pp. 5769–5779 (2017)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
18. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. (2017). arXiv Preprint
19. Jeong, S., Solenthaler, B., Pollefeys, M., Gross, M., et al.: Data-driven fluid simulations using regression forests. ACM Trans. Graph. **34**(6), 199 (2015)
20. Kalchbrenner, N., Oord, A., Simonyan, K., Danihelka, I., Vinyals, O., Graves, A., Kavukcuoglu, K.: Video pixel networks. (2016). arXiv:1610.00527
21. Kim, B., Liu, Y., Llamas, I., Rossignac, J.R.: Flowfixer: Using bfecc for fluid simulation. Technical report, Georgia Institute of Technology (2005)
22. Kim, B., Azevedo, V.C., Thuerey, N., Kim, T., Gross, M., Solenthaler, B.: Deep fluids: a generative network for parameterized fluid simulations. (2018). arXiv:1806.02071
23. Kim, J., Kwon Lee, J., Mu Lee, K.: Accurate image super-resolution using very deep convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1646–1654 (2016)
24. Kim, T., Thürey, N., James, D., Gross, M.: Wavelet turbulence for fluid simulation. In: ACM Trans. Graph. (2008). https://doi.org/10.1145/1360612.1360649
25. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. (2013). arXiv:1312.6114
26. Koshizuka, S., Oka, Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluid. Nucl. Sci. Eng. **123**(3), 421–434 (1996)
27. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in neural information processing systems, vol. 25, pp. 1097–1105 (2012)
28. Lee, A.X., Zhang, R., Ebert, F., Abbeel, P., Finn, C., Levine, S.: Stochastic adversarial video prediction. (2018). arXiv:1804.01523
29. Li, X., Liu, L., Wu, W., Liu, X., Wu, E.: Dynamic bfecc characteristic mapping method for fluid simulations. Vis. Comput. **30**(6–8), 787–796 (2014)
30. Long, Z., Lu, Y., Ma, X., Dong, B.: Pde-net: learning pdes from data. (2017). arXiv:1710.09668
31. Lotter, W., Kreiman, G., Cox, D.: Unsupervised learning of visual structure using predictive generative networks. (2015). arXiv:1511.06380
32. Maggioni, M., Boracchi, G., Foi, A., Egiazarian, K.: Video denoising using separable 4d nonlocal spatiotemporal transforms. In: Image processing: algorithms and systems IX, International Society for Optics and Photonics, vol. 7870, p. 787003 (2015)
33. Maggioni, M., Boracchi, G., Foi, A., Egiazarian, K.: Video denoising, deblocking, and enhancement through separable 4-d nonlocal spatiotemporal transforms. IEEE Trans. Image Process. **21**(9), 3952–3966 (2012). https://doi.org/10.1109/TIP.2012.2199324
34. Mathieu, M., Couprie, C., LeCun, Y.: Deep multi-scale video prediction beyond mean square error. (2015). arXiv:1511.05440. https://doi.org/10.1109/TIP.2012.2199324
35. McAdams, A., Sifakis, E., Teran, J.: A parallel multigrid Poisson solver for fluids simulation on large grids. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on computer animation, Eurographics Association, pp. 65–74 (2010)
36. Meng, Z., Weixin, S., Yinling, Q., Hanqiu, S., Jing, Q., Heng, P.A.: Vortex filaments in grids for scalable, fine smoke simulation. IEEE Comput. Graph. Appl. **35**(6), 60–68 (2015)
37. Mercier, O., Beauchemin, C., Thuerey, N., Kim, T., Nowrouzezahrai, D.: Surface turbulence for particle-based liquid simulations. ACM Trans. Graph. **34**(6), 202 (2015)
38. Mirza, M., Osindero, S.: Conditional generative adversarial nets. (2014). arXiv:1411.1784

39. Okabe, M., Anjyor, K., Onai, R.: Creating fluid animation from a single image using video database. Comput. Graph. Forum. **30**(7), 1973–1982 (2011). https://doi.org/10.1111/j.1467-8659.2011.02062.x
40. Okabe, M., Dobashi, Y., Anjyo, K., Onai, R.: Fluid volume modeling from sparse multi-view images by appearance transfer. ACM Trans. Graph. **34**(4), 93 (2015)
41. Pfaff, T., Thuerey, N., Gross, M.: Lagrangian vortex sheets for animating fluids. ACM Trans. Graph. **31**(4), 112 (2012)
42. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. (2015). arXiv:1511.06434
43. Ranzato, M., Szlam, A., Bruna, J., Mathieu, M., Collobert, R., Chopra, S.: Video (language) modeling: a baseline for generative models of natural videos. (2014). arXiv:1412.6604
44. Rasmussen, N., Nguyen, D.Q., Geiger, W., Fedkiw, R: Smoke simulation for large scale phenomena. ACM Trans. Graph. **22**(3), 703–707 (2003). https://doi.org/10.1145/882262.882335
45. Saito, M., Matsumoto, E., Saito, S.: Temporal generative adversarial nets with singular value clipping. In: IEEE international conference on computer vision (ICCV), pp. 2830–2839 (2017)
46. Selle, A., Rasmussen, N., Fedkiw, R.: A vortex particle method for smoke, water and explosions. ACM Trans. Graph. **24**(3), 910–914 (2005). https://doi.org/10.1145/1073204.1073282
47. Selle, A., Fedkiw, R., Kim, B., Liu, Y., Rossignac, J.: An unconditionally stable maccormack method. J. Sci. Comput. **35**(2–3), 350–371 (2008)
48. Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.-C.: Convolutional lstm network: a machine learning approach for precipitation nowcasting. Adv. Neural Inf. Process. Syst. **28**, 802–810 (2015)
49. Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: The IEEE conference on computer vision and pattern recognition (CVPR), vol. 3, p. 6 (2017)
50. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. (2014). arXiv:1409.1556
51. Srivastava, N., Mansimov, E., Salakhudinov, R.: Unsupervised learning of video representations using lstms. In: International conference on machine learning, pp. 843–852 (2015)
52. Stam, J.: Stable fluids. In: Proceedings of the 26th annual conference on computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., pp. 121–128 (2015)
53. Thuerey, N.: Interpolations of smoke and liquid simulations. ACM Trans. Graph. **36**(1), 3 (2017)
54. Thuong, H.: Video background HD—smoke HD—style proshow. (2014). https://www.youtube.com/watch?v=B6H34IccWks
55. Tompson, J., Schlachter, K., Sprechmann, P., Perlin, K.: Accelerating Eulerian fluid simulation with convolutional networks. (2016). arXiv:1607.03597
56. Um, K., Hu, X., Thuerey, N.: Liquid splash modeling with neural networks. Comput. Graph. Forum. **37**(8), 171–182 (2018). https://doi.org/10.1111/cgf.13522
57. Wang, C., Wang, C., Qin, H., Zhang, T.: Video-based fluid reconstruction and its coupling with sph simulation. Vis. Comput. **33**(9), 1211–1224 (2017)
58. Wen, J., Ma, H.: Real-time smoke simulation based on vorticity preserving lattice Boltzmann method. Vis.Comput. **35**(9), 1279–1292 (2019)
59. Wiewel, S., Becher, M., Thuerey, N.: Latent-space physics: Towards learning the temporal evolution of fluid flow. (2018). arXiv:1802.10123
60. Xie, Y., Franz, E., Chu, M., Thuerey, N.: tempogan: a temporally coherent, volumetric gan for super-resolution fluid flow. (2018). arXiv:1801.09710
61. Yoon, J.C., Kam, H.R., Hong, J.M., Kang, S.J., Kim, C.H.: Procedural synthesis using vortex particle method for fluid simulation. Comput. Graph. Forum. **28**(7), 1853–1859 (2009). https://doi.org/10.1111/j.1467-8659.2009.01563.x
62. Zhang, X., Bridson, R., Greif, C.: Restoring the missing vorticity in advection-projection fluid solvers. ACM Trans. Graph. **34**(4), 52 (2015)
63. Zhu, Y., Bridson, R.: Animating sand as a fluid. ACM Trans. Graph. **24**(3), 965–972 (2005)



Jinghuan Wen received the B.Eng. degree in Electronic Engineering from Tsinghua University, Beijing, China, in 2014, where he is currently pursuing the Ph.D degree. His research interests include computer graphics, physically based simulation, and deep learning.

Huimin Ma received the M.S. and Ph.D. degrees in Mechanical Electronic Engineering from Beijing Institute of Technology, Beijing, China, in 1998 and 2001, respectively. She is an associate professor in the Department of Electronic Engineering of Tsinghua University and the director of 3D Image Simulation Lab. She worked as an visiting scholar in University of Pittsburgh in 2011. She is also the executive director and the vice secretary general of China Society of Image and Graphics. Her research and teaching interests include 3D object recognition and tracking, system modeling and simulation, psychological base of image cognition.

Xiong Luo received the Ph.D. degree in computer applied technology from Central South University, Changsha, China, in 2004. He is currently a Professor with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. His current research interests include machine learning, computer vision, and computational intelligence. He has published extensively in his areas of interest in several journals, such as IEEE Transactions on Industrial Informatics, IEEE Transactions on Human-Machine Systems, and IEEE Transactions on Network Science and Engineering.