COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2016; **27**:415–424
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1695

SPECIAL ISSUE PAPER

Data-driven projection method in fluid simulation

Cheng Yang, Xubo Yang* and Xiangyun Xiao

School of Software, Shanghai Jiao Tong University, Shanghai, China

ABSTRACT

Physically based fluid simulation requires much time in numerical calculation to solve Navier–Stokes equations. Especially in grid-based fluid simulation, because of iterative computation, the projection step is much more time-consuming than other steps. In this paper, we propose a novel data-driven projection method using an artificial neural network to avoid iterative computation. Once the grid resolution is decided, our data-driven method could obtain projection results in relatively constant time per grid cell, which is independent of scene complexity. Experimental results demonstrated that our data-driven method drastically speeded up the computation in the projection step. With the growth of grid resolution, the speed-up would increase strikingly. In addition, our method is still applicable in different fluid scenes with some alterations, when computational cost is more important than physical accuracy. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS

artificial neural network; data-driven; grid-based fluid simulation; projection step

Supporting information may be found in the online version of this article.

***Correspondence**

Xubo Yang, School of Software, Shanghai Jiao Tong University, Shanghai, China.
E-mail: yangxubo@sjtu.edu.cn

1. INTRODUCTION

Physically based fluid simulation has been considered as one of the most important topics in computer graphics research. Navier–Stokes (N-S) equations need to be solved in the simulation process. Because N-S equations are nonlinear partial differential equations, many numerical simulation methods, such as Lagrangian methods [1] and Eulerian methods [2,3], are used to discretize them. In the field of high-resolution fluid simulation, Eulerian methods are widely used because they are more accurate in reconstructing and rendering surface.

To make the fluids incompressible, Eulerian methods need to solve the Poisson equation in the projection step. Because the Poisson equation needs to be solved iteratively by traditional numerical methods, the projection step is usually more time-consuming than other steps in the whole simulation process. It will cost a large amount of computing resource [4] and make it inconvenient when tiny adjustments occurred in fluid simulation. To speed up the projection step, many efficient numerical methods such as preconditioned conjugate gradient (PCG) method [5] and composite grid method [6] were proposed in fluid simulation. These methods still depend on iterative computation. Because PCG method has a good ability of convergence and is easy to implement, it becomes a common method in the projection step.

Machine learning-related methods are widely used in recent years. In this paper, we propose to use machine learning in the projection step to solve the Poisson equation instead of using iterative computation. The artificial neural network is an efficient machine learning technique and is widely used in classification and numerical fitting problems [7]. A naive idea of using the artificial neural network in grid-based fluid simulation is to obtain the density field in the next frame by importing the density field in the current frame into the artificial neural network. However, the relationship of density fields between frames is unstable, which is easily influenced by the time step, external forces, and other factors. In contrast, the relationship between input and output data in the projection step is comparatively stable, so it is barely influenced in different fluid scenes. Thus, we propose a novel data-driven projection method using the artificial neural network, in order to greatly speed up the projection step and the whole simulation process.

In the following sections, we first briefly introduced the traditional grid-based fluid simulation framework. Then, we defined a feature vector as the input layer in the artificial neural network, while the output layer is the pressure value in the next frame. Finally, we constructed an artificial neural network and trained it as a Poisson equation solver. Our data-driven projection method has the following features:

Figure 1. The simulation results using our data-driven projection method (resolution: 192 × 256 × 192). It could speed up more than 10 times than preconditioned conjugate gradient method. Experimental results suggested the effectiveness of our method to time-critical fluid simulation.

- *Fast computation.* The computing time only depends on grid resolution, and iterative computation can be avoided when the artificial neural network is trained completely. In addition, the speed-up ratio will continuously increase with the growth of grid resolution.
- *Realistic visual effects.* Artificial neural networks can restore the nonlinear relationship between input and output data in the projection step, so the projection results of our method are visually acceptable (Figure 1).
- *Satisfactory extrapolation ability.* The relationship between input and output data in the Poisson equation is comparatively stable. So our data-driven method is still applicable in various fluid scenes with some alterations.
- *Generality.* Our data-driven projection method is general enough to be integrated into other grid-based related fluid simulation methods, such as vorticity confinement [3] and synthetic turbulence [8].

2. RELATED WORK

Stam *et al.* [2,3] showed that Eulerian methods could stably solve the N-S equations in fluid simulation. Compared with Lagrangian methods [1], Eulerian methods are more stable and more accurate to reconstruct and render the fluid surface. However, the projection step in Eulerian methods needs to solve the Poisson equation, which costs a large amount of computing resource [4]. So many research work is trying to reduce required computing resources and speed up fluid simulation.

One direct acceleration method is using more efficient numerical methods to solve the Poisson equation. Bridson *et al.* [5] used PCG method in grid-based fluid simulation, which could greatly reduce the number of iterations while solving large-scale sparse linear equations. It could replace the widely used Gauss–Seidel iterative method [9]. McAdams *et al.* [10] proposed multi-grid as a preprocessing step of conjugate gradient method. The acceleration effect is obvious when the simulation scene is on a large scale. Another often used idea is composite grid method. The purpose of composite grid method is to allocate more computing resource in important area and finally speed up the simulation on the premise of guaranteeing visual effect. Losasso *et al.* [6] implemented the octree data structure to create fluid grids. They used fine grids in detailed area and coarse grids in steady area, in order to reduce computational cost. Zhu *et al.* [11] set fine grids in the source position where fluid was generated. The issues of composite grid method are poor robustness in different fluid scenes. Procedural detail synthesis method can effectively increase turbulent details in re-simulation process [8,12]. It can effectively improve the visual effect in a short time. However, these methods cannot avoid solving the divergence-free condition in the Poisson equation with iterative steps and time-consuming computation.

Data-driven methods have been widely used in various physically based simulation. In fluid simulation, the most common idea is to transform the solving process into low-dimensional space [13,14]. With the help of dimension reduction methods like principal component analysis, complex high-dimensional fluid dynamic equations can be transformed into low-dimensional space and become fast to solve. At the cost that much visual effects are missed, the calculation process is sped up. However, this idea cannot avoid computing iteratively, and it still has the potential for further acceleration. Another common data-driven idea is interpolation [15]. Using existing fluid simulation data, interpolation methods can rapidly predict simulation data without directly solving complex equations. Jeong *et al.* [16] regarded Lagrangian fluid simulation process as a regression problem. In the preprocessing step, they trained regression forests by historical velocity and density data. After training completely, the trained regression forests could rapidly obtain velocity in the next frame according to the existing feature vector. However, the regression method was only used in particle-based fluid simulation.

The artificial neural network is a self-adaptive computational method inspired by biological neural network [7]. By increasing hidden layers and neurons, artificial neural networks can efficiently build a model to describe the relationship or explore hidden patterns between input and output data. The outputs of artificial neural networks

DOI: 10.1002/cav

depend on internal structure, connection weight, activation function, and bias value. Artificial neural networks have played an important role in the fields of face recognition [17], financial forecast [18], image classification [19], and so on. In this paper, we use the trained artificial neural network as projection solver, which could avoid iterative steps and time-consuming computation.

3. GRID-BASED FLUID DYNAMICS

In this section, we will introduce traditional projection methods. The common symbols used in this paper are shown in Table I.

Physically based fluid simulation process can be described by famous N-S equation. N-S equations consist of a set of differential equations. Before describing traditional projection methods, we briefly introduce N-S equations:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=-\nabla p+\kappa\nabla\cdot\nabla\mathbf{u}+\rho\mathbf{f} \tag{1}$$

$$\nabla\cdot\mathbf{u}=0 \tag{2}$$

Equation (1) is used to describe the process of fluid motion, and Equation (2) is used to make fluids incompressible. Stam [2] divided the whole solving process into four steps: advection step, external force step, diffusion step, and projection step. As for inviscid fluid, there are only three steps without diffusion step. The basic Eulerian framework of inviscid fluid is shown in Algorithm 1. In advection step, fluid grids just change their positions. External force step is easy to implement and only needs to change acceleration per grid. In this framework, the projection step is the most time-consuming step, which aims at getting pressure in every grid.

Table I. Common symbols used in this paper.

$\boldsymbol{x}_{i,j,k}$	Eulerian grid position at (*i, j, k*)
$\rho(\boldsymbol{x})$	Density in grid
$\mathbf{u}_n^M(\boldsymbol{x})$	Velocity in grid on step *M* in the *n*th frame
$p_n(\boldsymbol{x})$	Pressure in grid in the *n*th frame
$\mathbf{f}$	External force
κ	Viscosity factor
Δt	Time step between two frames
h	Width of the grid cell
$\boldsymbol{\beta}_n(\boldsymbol{x})$	Feature vector in grid in the *n*th frame
n_l	Number of layers in neural network
α	Learning rate in neural network
$w_{ij}^{(l)}$	Weight on the connection from the *j*th neuron in layer *l* to the *i*th neuron in layer (*l*+1)
$bias_i^{(l)}$	Bias from bias neuron in layer *l* to the *i*th neuron in layer (*l*+1)
$\delta_i^{(l)}$	Residual in the *i*th neuron in layer *l*
$z_i^{(l)}$	Sum in the *i*th neuron in layer *l*
$a_i^{(l)}$	Output in the *i*th neuron in layer *l*

Algorithm 1 Eulerian solution framework

Input:
The velocity in the *n*th frame, $\mathbf{u}_n$;
The external force per frame, $\mathbf{f}$;
Output:
The velocity in the next frame, $\mathbf{u}_{n+1}$;
1: **for** each grid position $\boldsymbol{x}$ **do**
2: $\mathbf{u}_n^A(\boldsymbol{x})=Advect(\mathbf{u}_n(\boldsymbol{x}))$;
3: $\mathbf{u}_n^B(\boldsymbol{x})=\mathbf{u}_n^A(\boldsymbol{x})+\Delta t\mathbf{f}$;
4: $\mathbf{u}_{n+1}(\boldsymbol{x})=Project(\mathbf{u}_n^B(\boldsymbol{x}))$;
5: **end for**
6: **return** $\mathbf{u}_{n+1}$;

In this paper, we use marker-and-cell grids [20] to discretize the solver space of N-S equations. After discretization, the velocity at position $\boldsymbol{x}$ can be updated by the pressure in the projection step in Algorithm 1:

$$\mathbf{u}_{n+1}(\boldsymbol{x})=\mathbf{u}_n^B(\boldsymbol{x})-\Delta t\frac{1}{\rho}\nabla p_n \tag{3}$$

Thus, the main target in the projection step is to solve pressure per grid in the current frame. Then, the velocity per grid can be updated by solving the pressure in neighboring grids. According to Equation (2), Equation (3) can also be expressed as the Poisson equation:

$$\nabla\cdot\mathbf{u}_n^B(\boldsymbol{x})=\Delta t\frac{1}{\rho}\nabla\cdot\nabla p_n \tag{4}$$

Given the velocity field after external force step, and advection step, this Poisson Equation (4) can be discretized as

$$\frac{\Delta t}{\rho}\begin{pmatrix}\dfrac{\begin{array}{r}6p_n(\boldsymbol{x}_{i,j,k})\\ -p_n(\boldsymbol{x}_{i+1,j,k})-p_n(\boldsymbol{x}_{i,j+1,k})\\ -p_n(\boldsymbol{x}_{i,j,k+1})-p_n(\boldsymbol{x}_{i-1,j,k})\\ -p_n(\boldsymbol{x}_{i,j-1,k})-p_n(\boldsymbol{x}_{i,j,k-1})\end{array}}{h^2}\end{pmatrix}=\begin{pmatrix}\dfrac{\begin{array}{r}\mathbf{u}_n^B(\boldsymbol{x}_{i+1,j,k})-\mathbf{u}_n^B(\boldsymbol{x}_{i-1,j,k})\\ +\mathbf{u}_n^B(\boldsymbol{x}_{i,j+1,k})-\mathbf{u}_n^B(\boldsymbol{x}_{i,j-1,k})\\ +\mathbf{u}_n^B(\boldsymbol{x}_{i,j,k+1})-\mathbf{u}_n^B(\boldsymbol{x}_{i,j,k-1})\end{array}}{h}\end{pmatrix} \tag{5}$$

Combining all linear equations in every grid after discretization, we rewrote them in matrix form as follows:

$$\boldsymbol{A}\boldsymbol{p} = \boldsymbol{d} \tag{6}$$

where $\boldsymbol{A}$ is a large-scale sparse matrix that only depends on the boundary condition and the discretization form, $\boldsymbol{p}$ is a vector containing the pressure in every grid, and $\boldsymbol{d}$ is a vector containing the velocity gradients. Bridson *et al.* [5] used PCG method to dramatically reduce iterations for solving this large-scale sparse system of equations. Generally, the projection step in Eulerian method would consume about 80–90% computing resources of the whole process.

4. DATA-DRIVEN PROJECTION METHOD

Our data-driven projection method in grid-based fluid simulation framework is shown in Figure 2. The artificial neural network is used to be the solver in our method, and its input layer is the feature vector of projection step. The feature vector needs to include all known variables, which influence the results of Poisson equation. Trained by the training samples from basic simulation scenes, our method can rapidly solve the Poisson equation according to incoming feature vectors.

In the following subsections, we define the feature vector and introduce the artificial neural network used in our method.

4.1. Feature Vector

We define R as the box $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ and each grid at position $\boldsymbol{x}_{i,j,k} \in R$. Then let $\boldsymbol{Q}$ be the matrix that is composed of the point-wise property q per grid, such as pressure, boundary, velocity, and so on:

$$q(\boldsymbol{x}_{i,j,k}) \in \boldsymbol{Q}(R), \boldsymbol{x}_{i,j,k} \in R \tag{7}$$

Many grid-based fluid simulation processes were generated by PCG method in the projection step, so PCG method is used as the ground truth of our data-driven method. The key to speed up PCG method is the choice of initial value before iterations. Because the gap of pressure between two sequential frames is small, the pressure in last frame is set to be initial value. At the same time, PCG method also depends on the boundary condition. Neumann boundary condition is used in this paper. The function $o(\boldsymbol{x})$ represents the boundary condition:

$$o(\boldsymbol{x}_{i,j,k}) = \begin{cases} 1 & \text{if obstacle at } \boldsymbol{x}_{i,j,k}. \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

According to PCG method, the solving process in the projection step can be regarded as a function named *PCG*:

$$\boldsymbol{P}_n(R) = PCG(\boldsymbol{P}_{n-1}(R), \nabla \mathbf{U}_n(R), \boldsymbol{O}_n(R)) \tag{9}$$

Like Equation (7), we represent that $p_n(\boldsymbol{x}_{i,j,k}) \in \boldsymbol{P}_n(R)$, $\nabla \mathbf{u}_n(\boldsymbol{x}_{i,j,k}) \in \nabla \mathbf{U}_n(R)$, $o_n(\boldsymbol{x}_{i,j,k}) \in \boldsymbol{O}_n(R)$. If directly training the matrices $\boldsymbol{P}_{n+1}(R)$, $\nabla \mathbf{U}_n(R)$ and $\boldsymbol{O}_n(R)$, the required memory is unacceptable. It is necessary to solve the equation in each grid. To implement PCG method, Equation (9) should be transformed as Equation (10), where function *pcg* is a vector form of *PCG*:

$$\begin{aligned} p_n(\boldsymbol{x}_{i,j,k}) = pcg(&p_{n-1}(\boldsymbol{x}_{i,j,k}), \\ &p_{n-1}(\boldsymbol{x}_{i-1,j,k}), \nabla \mathbf{u}_n(\boldsymbol{x}_{i-1,j,k}), o_n(\boldsymbol{x}_{i-1,j,k}), \\ &p_{n-1}(\boldsymbol{x}_{i+1,j,k}), \nabla \mathbf{u}_n(\boldsymbol{x}_{i+1,j,k}), o_n(\boldsymbol{x}_{i+1,j,k}), \\ &p_{n-1}(\boldsymbol{x}_{i,j-1,k}), \nabla \mathbf{u}_n(\boldsymbol{x}_{i,j-1,k}), o_n(\boldsymbol{x}_{i,j-1,k}), \\ &p_{n-1}(\boldsymbol{x}_{i,j+1,k}), \nabla \mathbf{u}_n(\boldsymbol{x}_{i,j+1,k}), o_n(\boldsymbol{x}_{i,j+1,k}), \\ &p_{n-1}(\boldsymbol{x}_{i,j,k-1}), \nabla \mathbf{u}_n(\boldsymbol{x}_{i,j,k-1}), o_n(\boldsymbol{x}_{i,j,k-1}), \\ &p_{n-1}(\boldsymbol{x}_{i,j,k+1}), \nabla \mathbf{u}_n(\boldsymbol{x}_{i,j,k+1}), o_n(\boldsymbol{x}_{i,j,k+1})) \end{aligned} \tag{10}$$

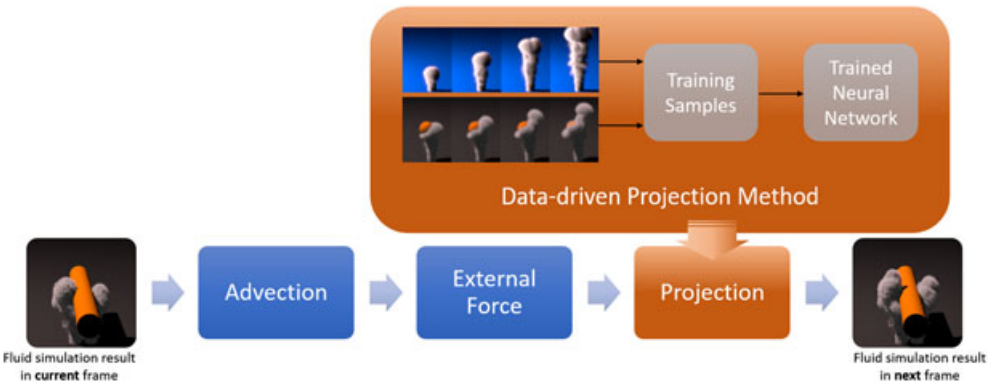


Figure 2. Our data-driven projection method does not change other steps in traditional grid-based fluid simulation framework. Trained by basic sample data, our data-driven projection method could solve the Poisson equation much faster than traditional projection methods.

DOI: 10.1002/cav

According to Equation (10), we define a feature vector for our data-driven method, a 19-dimensional vector $\boldsymbol{\beta}_n(x)$:

$$\begin{aligned}\boldsymbol{\beta}_n(\boldsymbol{x}_{i,j,k}) = [&p_{n-1}(\boldsymbol{x}_{i,j,k}),\\ &p_{n-1}(\boldsymbol{x}_{i-1,j,k}), \nabla\mathbf{u}_n(\boldsymbol{x}_{i-1,j,k}), o_n(\boldsymbol{x}_{i-1,j,k}),\\ &p_{n-1}(\boldsymbol{x}_{i+1,j,k}), \nabla\mathbf{u}_n(\boldsymbol{x}_{i+1,j,k}), o_n(\boldsymbol{x}_{i+1,j,k}),\\ &p_{n-1}(\boldsymbol{x}_{i,j-1,k}), \nabla\mathbf{u}_n(\boldsymbol{x}_{i,j-1,k}), o_n(\boldsymbol{x}_{i,j-1,k}),\\ &p_{n-1}(\boldsymbol{x}_{i,j+1,k}), \nabla\mathbf{u}_n(\boldsymbol{x}_{i,j+1,k}), o_n(\boldsymbol{x}_{i,j+1,k}),\\ &p_{n-1}(\boldsymbol{x}_{i,j,k-1}), \nabla\mathbf{u}_n(\boldsymbol{x}_{i,j,k-1}), o_n(\boldsymbol{x}_{i,j,k-1}),\\ &p_{n-1}(\boldsymbol{x}_{i,j,k+1}), \nabla\mathbf{u}_n(\boldsymbol{x}_{i,j,k+1}), o_n(\boldsymbol{x}_{i,j,k+1})]\end{aligned} \tag{11}$$

4.2. Artificial Neural Network

The artificial neural network is a computational method that contains many neurons. Every neuron is a basic data processing unit. The output of neural network depends on connection modes, weights, and activation functions.

Neurons. The basic data processing unit in neural network is a neuron. The connection between two neurons is the value of the product of transmitted signal and connection weight. The neuron aggregates the values of the products from connected neurons in last layer then puts the sum into activation function f, and gets the final output signal a:

$$a_i^{(l)} = f\left(z_i^{(l)}\right) \tag{12}$$

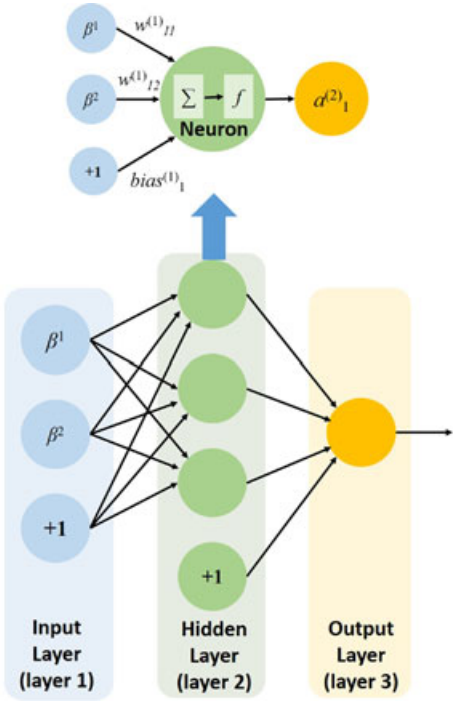


Figure 3. Example of three-layer neural network and one of its neuron.

The activation function f used in this paper was *sigmoid*() [21], which is used to add nonlinear factors. The function of an individual neuron is to split a high-dimensional space into two parts. To be convenient for explanation, we give a simple three-layer neural network in Figure 3 as an example.

The input of this network is a two-dimensional feature vector $\boldsymbol{\beta}_{sample} = (\beta^1, \beta^2)$. The left layer is input layer and the right layer is output layer. The hidden layer is in the middle, and it can contain more than one layer. And the more hidden layers, the more accurate relationship between input and output data can be described, but the training time will tremendously increase. The **+1** neuron is bias neuron, and its output signal is constant to 1.

In our data-driven projection method, the 19-dimensional feature vector $\boldsymbol{\beta}$ is set as input layer. In other words, our neural network has 20 input neurons (including the bias neuron) in input layer. And the sole neuron in output layer is the pressure in the next frame.

Forward propagation. In Figure 3, n_l=3. Let the feature vector $\boldsymbol{\beta}_{sample}$ be the input of network; we can start forward propagation first:

$$\begin{cases} a_1^{(2)} = f\left(w_{11}^{(1)}\beta_1 + w_{12}^{(1)}\beta_2 + bias_1^{(1)}\right) \\ a_2^{(2)} = f\left(w_{21}^{(1)}\beta_1 + w_{22}^{(1)}\beta_2 + bias_2^{(1)}\right) \\ a_3^{(2)} = f\left(w_{31}^{(1)}\beta_1 + w_{32}^{(1)}\beta_2 + bias_3^{(1)}\right) \end{cases} \tag{13}$$

$$\begin{aligned}&\text{output}(\boldsymbol{\beta}_{sample}) = a_1^{(3)} = \\ &f\left(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + bias_1^{(2)}\right)\end{aligned} \tag{14}$$

The forward propagation is finished after Equations (13) and (14). Then, we will obtain the output value from the neural network. It is important to note that the *output values* (obtained by neural network) here may not be

Algorithm 2 Detailed back propagation

Input:
Feature vectors $\boldsymbol{\beta}$ in training samples;
Target values in training samples;
Output:
Weights and biases;
while $\sum \delta_i^{(n_l)} <$ threshold **do**
  for each training sample $\boldsymbol{\beta}$ **do**
    ForwardPropagation($\boldsymbol{\beta}$);
    Compute $\delta_i^{(n_l)}$ in output layer;
    $\delta_i^{(l)} = \left(\sum w_{ji}^{(l)}\delta_j^{(l+1)}\right) f'\left(z_i^{(l)}\right), l = n_l-1, n_l-2, \ldots, 2;$
    Update weights and biases:
    $w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha a_j^{(l)}\delta_i^{(l+1)}$
    $bias_i^{(l)} = bias_i^{(l)} - \alpha\delta_i^{(l+1)};$
  end for
end while
return Trained weights and biases;

equivalent to the *target values* (actual output value from training samples).

Back propagation. The most complicated step in neural network is how to adjust weights and biases. We can use the back propagation algorithm to adjust these values. Given a training sample, run forward propagation for one time and compute the residuals in every neuron. Then, we could use residuals to update weights and biases by gradient descent method [22]. In output layer, residual $\delta_i^{(n_l)}$ could be directly calculated as the square of difference values between the *target values* and *output values*. However, it cannot be directly calculated in hidden layer, it can only be transmitted from the next layer. The detailed implementation pseudo-code of back propagation algorithm is shown in Algorithm 2.

Through acquiring feature vectors (input layer) and pressures (output layer) in every grid within selected frames as training samples, neural network can be trained to replace other Poisson equation solvers. Once the neural network is trained completely, we input a feature vector per grid into this neural network and run forward propagation for one time; then, we obtain the solution of Poisson equation. It can completely avoid iterative steps and time-consuming computation.

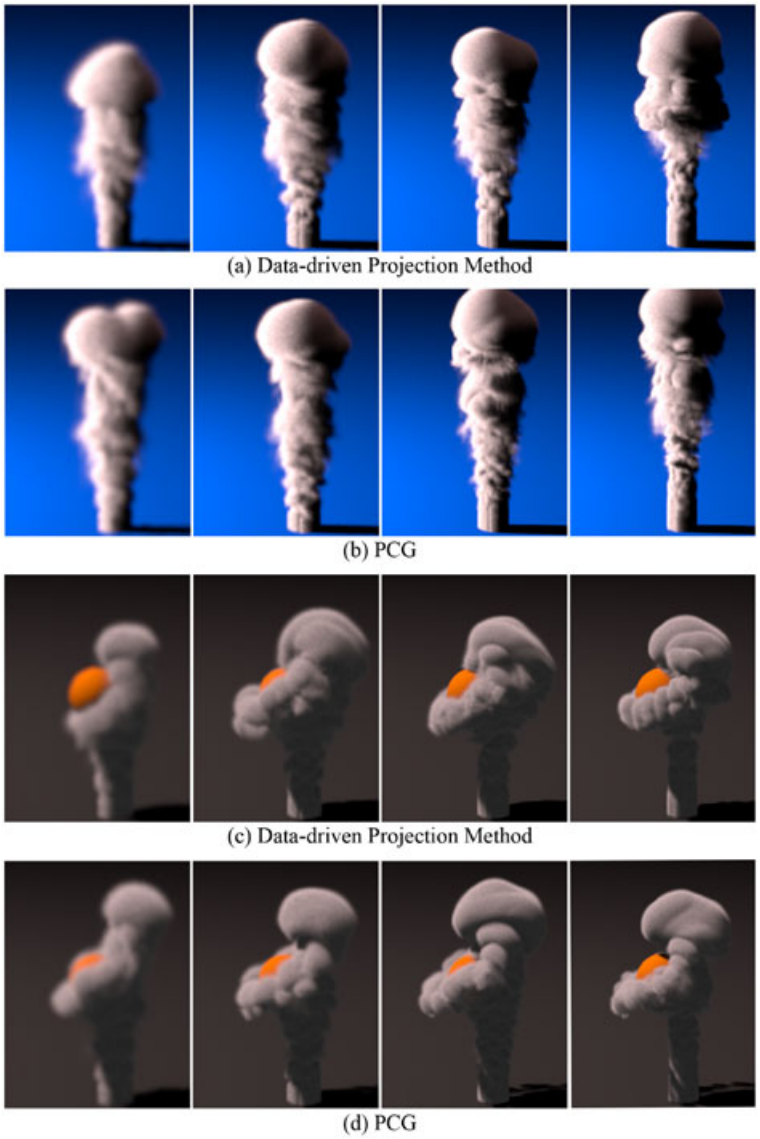


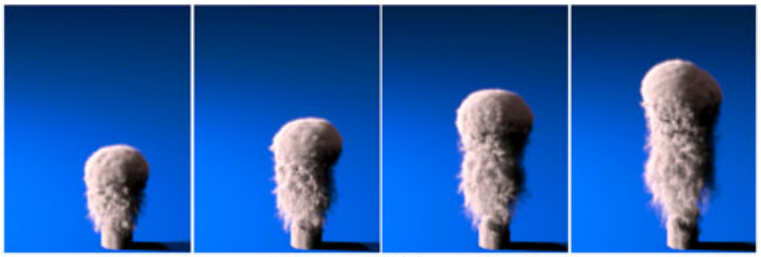
Figure 4. Comparison with the simulation results using preconditioned conjugate gradient (PCG) method in different grid resolutions (from left to right, 48 × 64 × 48, 96 × 128 × 96, 192 × 256 × 192, and 384 × 512 × 384). Our method obtained similar results with some differences.

DOI: 10.1002/cav

Table II. Computing time in the projection step using our method and PCG method per frame and the speed-ups using our data-driven method.

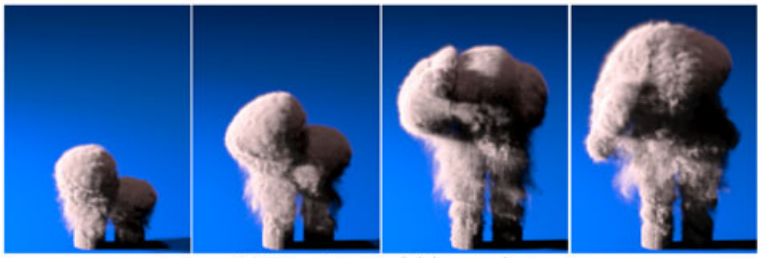
Scene	Resolution	PCG (*s*)	Data-driven (*s*)	Speed-up
Rising smoke (Figure 4(a) and (b))	48 × 64 × 48	0.282	0.063	4.476
	96 × 128 × 96	3.938	0.515	7.647
	192 × 256 × 192	52.078	4.924	10.576
	384 × 512 × 384	761.734	52.151	14.606
With sphere (Figure 4(c) and (d))	48 × 64 × 48	0.273	0.062	4.403
	96 × 128 × 96	3.842	0.518	7.417
	192 × 256 × 192	52.175	4.882	10.687
	384 × 512 × 384	780.071	52.097	14.973

Our machine learning method successfully solved the Poisson equation in grid-based fluid simulation and showed the potential in time-critical simulation, especially in high-resolution scenes. PCG, preconditioned conjugate gradient.

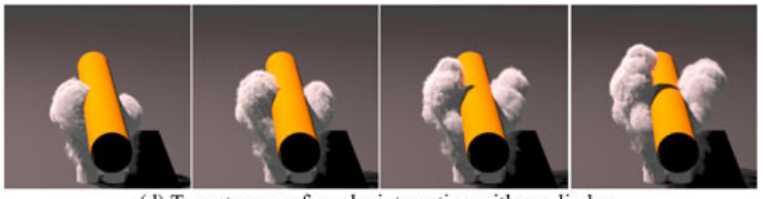


(a) A stream of rising smoke

(b) A stream of smoke interacting with a sphere



(c) Two streams of rising smoke



(d) Two streams of smoke interacting with a cylinder

Figure 5. The simulation results using our data-driven projection method in different simulation scenes, which added vorticity confinement and synthetic turbulence.

5. EXPERIMENTS

The main configuration of our experimental platform is as follows: Intel(R) Core i7-4790K CPU at 4.00 GHz, 16.0 GB memory.

Data acquisition. The training samples as ground truth were generated by PCG method. The training input data (feature vector $\boldsymbol{\beta}$) and training target data (pressure p after the projection step by PCG method) were obtained by two basic simulation scenes. The first scene is composed of a stream of rising smoke, and the second scene is composed of a stream of smoke and a sphere obstacle. These scenes were the basis of other fluid simulation scenes like interacting, appearing, or disappearing obstacles. Each training video contained 147 000 grids per frame, and training data were obtained in 40 randomly selected frames of the whole simulation process. So we obtained 11.8 million training samples, which contained input data and target data.

Neural network training. As for machine learning, test set and cross-validation set are as important as training set. Test set is used to evaluate extrapolation ability of neural network and avoid over-fitting. Cross-validation set is used to correct training parameters [7].

The neural network has 20 neurons (including the bias neuron) in input layer and only one neuron in output layer. We randomly selected 15% of all training samples as cross-validation set, another 15% as test set, and the remaining 70% as training set for our neural network. Throughout our cross validation step and test step, when the number of hidden layer reached three and the number of neurons per hidden layer reached six, the nonlinear relationship between input layer and output layer could be relatively accurately described by neural network. In this paper, the neural network has three hidden layers and six neurons per hidden layer. After training for 31 hours, we obtained the final trained neural network. In addition, only the relationship of data and the numbers of training samples would effect the training time.

Comparison of results. We regarded the trained neural network as a Poisson equation solver. Input a feature vector to this network and run forward propagation for one time, we could obtain the solution of Poisson equation. In Figure 4, we compared simulation results using our method and PCG method in different grid resolutions. We also compared the computing time in Table II. In order to demonstrate the extrapolation ability of our method, we designed some complex simulation scenes in Figure 5(c) and (d). In addition, to demonstrate generality of our method, we implemented our data-driven method with vorticity confinement [3] and synthetic turbulence [8] in Figure 5.

Analysis of our method. Our method can speed up more than 10 times than previous methods in the projection step without GPU or multi-thread acceleration. And it does not influence the computing time in other steps. In Table II, we find that the speed-up of computing time depended on grid resolutions. With the growth of grid resolution, the speed-up will increase strikingly. So our method is particularly suitable for the simulation of large-scale fluid scenes.

Meanwhile, our simulation results keep the visually similarity with the results using traditional method, even with some degree of extrapolation. It is significantly helpful for the simulation where computing time matters more than physical accuracy. In addition, because our data-driven projection method is independent of other steps in traditional grid-based fluid simulation procedure, it is general enough to be applied to other grid-based related fluid simulation methods, such as vorticity confinement [3] and synthetic turbulence [8].

The main limitation of our method also exists in all machine learning-related methods: it does not have the ability of continuous learning, so our method cannot extrapolate the fluid scenes far outside the training samples.

6. CONCLUSIONS

In this paper, we propose a novel data-driven method that is supported by the artificial neural network to solve the Poisson equation in the projection step. We first defined a feature vector corresponding to velocity field, pressure field, and boundary condition. Then, we trained an artificial neural network by known feature vector per frame. We also explored the projection computing time in different grid resolutions and found that the speed-up of computing time was continuously increasing with the growth of grid resolution. The results demonstrated that our data-driven projection method has exhibited obvious superiority in large-scale fluid simulation.

In subsequent experiments, we need to prove the divergence-free condition in our method and compare our method with more acceleration methods. In the future, we will try parallel computing in our data-driven method to be more time-saving in training samples and computing projection results. Another direction is to apply online learning to current data-driven method to obtain a more powerful method with continuous learning ability, which could handle complex boundary conditions of arbitrary shape and bring more excellent extrapolation ability in various scenes.

ACKNOWLEDGEMENTS

This work is supported in part by the National Natural Science Foundation of China (nos. 61173105 and 61373085) and the National High Technology Research and Development Program of China (no. 2015AA016404).

REFERENCES

1. Müller M, Charypar D, Gross M. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, San Diego, USA, 2003, Eurographics Association; 154–159.

2. Stam J. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley Publishing Co., New York, USA, 1999; 121–128.
3. Fedkiw R, Stam J, Jensen HW. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, USA, 2001; 15–22.
4. Bolz J, Farmer I, Grinspun E, Schröoder P. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In *ACM Transactions on Graphics (TOG)*. ACM, New York, USA, 2003; 917–924.
5. Foster N, Fedkiw R. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, USA, 2001; 23–30.
6. Losasso F, Gibou F, Fedkiw R. Simulating water and smoke with an octree data structure. In *ACM Transactions on Graphics (TOG)*. ACM, New York, USA, 2004; 457–462.
7. Hagan MT, Demuth HB, Beale MH. *Neural Network Design*. Pws Pub.: Boston, 1996.
8. Kim T, Thürey N, James D, Gross M. Wavelet turbulence for fluid simulation. In *ACM Transactions on Graphics (TOG)*. ACM, New York, USA, 2008; 50.
9. Stam J. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*, 2003; 25.
10. 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, Madrid, Spain, 2010; 65–74.
11. Zhu B, Lu W, Cong M, Kim B, Fedkiw R. A new grid structure for domain extension. In *ACM Transactions on Graphics (TOG)*, New York, USA, 2013; 63:1–63:12.
12. Narain R, Sewall J, Carlson M, Lin M.C. Fast animation of turbulence using energy transport and procedural synthesis. In *ACM Transactions on Graphics (TOG)*. ACM, New York, USA, 2008; 166:1–166:8.
13. Treuille A, Lewis A, Popović Z. Model reduction for real-time fluids. In *ACM Transactions on Graphics (TOG)*. ACM, New York, USA, 2006; 826–834.
14. Ando R, Thürey N, Wojtan C. A dimension-reduced pressure solver for liquid simulations. In *Computer Graphics Forum (EUROGRAPHICS)*, Malden, USA, 2015; 10.
15. Raveendran K, Wojtan C, Thuerey N, Turk G. Blending liquids. In *ACM Transactions on Graphics (TOG)*, New York, USA, 2014; 137:1–137:10.
16. Ladický L, Jeong S, Solenthaler B. Data-driven fluid simulations using regression forests. In *ACM Transactions on Graphics (TOG)*, New York, USA, 2015; 199:1–199:9.
17. Rowley H, Baluja S, Kanade T. Neural network-based face detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, New York, USA, 1998; 23–38.
18. Kaastra I, Boyd M. Designing a neural network for forecasting financial and economic time series. In *Neurocomputing*, Amsterdam, The Netherlands, 1996; 215–236.
19. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. The MIT Press, Cambridge MA, US, 2012; 1097–1105.
20. Harlow FH, Welch JE. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. In *Physics of Fluids*, New York, USA, 1965; 2182–2189.
21. Han J, Moraga C. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *From Natural to Artificial Neural Computation*. Springer, Berlin, Germany, 1995; 195–201.
22. Avriel M. *Nonlinear Programming: Analysis and Methods*. Dover Pub.: New York, USA, 2003.

SUPPORTING INFORMATION

Supporting information may be found in the online version of this article.

AUTHORS' BIOGRAPHIES

Cheng Yang received his BEng degree in Computer Science at the School of Computer from Wuhan University in 2014. He is expected to receive his MS degree in Software Engineering on March 2017. His research focuses on computer graphics, machine learning, and so on.

Xubo Yang received his PhD degree in Computer Science from the State Key Lab of CAD&CG at Zhejiang University, in 1998. He is currently a Professor in the School of Software at Shanghai Jiao Tong University. His research interests include computer graphics, virtual reality, and human computer interaction.

Xiangyun Xiao received his MS degree in Computational Mathematics from Wuhan University last 2015 and is currently working toward his PhD degree in Software Engineering. His research is concerned with machine learning algorithms in fluid simulation, computational intelligence algorithms, and so on.

Comp. Anim. Virtual Worlds 2016; **27**:415–424 © 2016 John Wiley & Sons, Ltd.
DOI: 10.1002/cav