IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 27, 2020, accepted January 21, 2021, date of publication January 25, 2021, date of current version February 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054061

Learning Representation of Secondary Effects for Fire-Flake Animation

MYUNGJIN CHOI[1], JEONG A WI[2], (Graduate Student Member, IEEE), TAEHYEONG KIM[3], YOUNGBIN KIM[2], (Member, IEEE), AND CHANG-HUN KIM[4]

[1]Department of Computer and Radio Communications Engineering, Korea University, Seoul 02841, South Korea
[2]Graduate School of Advanced Imaging Science, Multimedia & Film, Chung-Ang University, Seoul 06974, South Korea
[3]CLO Virtual Fashion, Seoul 06039, South Korea
[4]Visual Information Processing-Dean of Department, Korea University, Seoul 02841, South Korea

Corresponding author: Chang-Hun Kim (chkim@korea.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] under Grant NRF-2019R1A2C1008244 and Grant NRF-2017R1A2B2005380.

ABSTRACT This paper proposes a new data-driven neural network-based fire-flake simulation model. Our model trains a neural network using precomputed fire simulation data. The trained neural network model generates fire flakes in appropriate locations and infers their velocity to make them appear natural to their surroundings. The neural network model consists of a fire-flake generator and a velocity modifier. The fire-flake generator uses the velocity, temperature, and density fields of the precomputed fire simulation as inputs to determine the locations at which natural fire flakes would be generated. The velocity modifier takes the velocity field of the precomputed fire simulation as input and infers the velocity of the generated fire flakes so that they appear natural relative to the flame motions and surroundings. Our method adopts a neural network to efficiently improve the fire-flake simulation, enhancing the performance while maintaining the visual quality. Our model is approximately three times faster than the traditional fire-flake model. In particular, our model is 30 times faster in the velocity modification step. Our method is also easier to implement than the existing physically based fire-flake simulation method and can reduce the time spent by artists and developers on their applications.

INDEX TERMS Fire-flake simulation, visual effect, visual simulation, machine learning, supervised learning.

I. INTRODUCTION

Fire flakes are small lightweight carbon-based particles that fall or float in the air during combustion of carbon-based materials. They occur because of external physical impacts or explosions caused by vapor or internal gas expansion. A realistic natural-looking representation of fire flakes requires depicting the dynamic animated effects of fire. However, fire-flake motion is complex and nearly unpredictable. Thus, artists attempting to represent fire flakes in a natural way exert a significant amount of time on trial and error.

Realizing an accurate simulation of fire flakes is challenging; however, their physical attributes can be inferred. Kim *et al.* [1] proposed a method to generate fire flakes that fit naturally in the surroundings. Abrupt changes in physical components are analyzed to generate fire flakes in appropriate locations, and the velocity field is analyzed to give the fire flakes a natural-looking velocity. Although the method proposed by Kim *et al.* can accelerate the work of artists, it does not work in real time because the grid-based simulation of the method requires an exponential increase in input data to generate higher-resolution and higher-quality fire flakes. Thus, the method has limited feasibility to obtain faster results.

Hence, machine learning has been applied to secondary-effect simulations, ranging from data-driven simulations based on neural networks trained with data generated by the existing simulation method to training on real-world physical phenomena, which are difficult to illustrate with algorithmic neural network modeling. Ultimately, these studies aimed to obtain faster and more realistic results; however, they do not entirely succeed.

Following recent research trends, we employed a neural network model to overcome the limitations of traditional

The associate editor coordinating the review of this manuscript and approving it for publication was Yunjie Yang.

fire-flake simulations [1]. Our method reduces the computational cost and maintains visual fidelity. Moreover, the proposed network is easy to implement, enabling users to easily employ their desired fire simulation model. Furthermore, it combines the existing method's multiple parameters into one parameter for better usability.

Our method consists primarily of two neural network components. First, the fire-flake generator analyzes the input velocity, temperature, and density fields to determine the locations at which fire flakes should be generated. This module calculates the probability of fire-flake generation in each simulation grid. Next, the velocity modifier analyzes the input velocity field and uses polynomial regression to infer the natural velocity of the generated fire flakes. Our neural network model was optimized for the fire-flake simulation, making it easy to implement, and it operates 2.4 times faster than the existing method.

II. RELATED WORK

A. FLUID SIMULATION

Fluid simulation is an important field of study and has been extensively investigated [2]–[4]. Stable fluids [5] are widely used in computer graphics and form a part of grid-based fluid simulation, which is a representative Eulerian method. The material point method, which is often used with the grid-based method, is used to depict complex fluid-like materials, such as snowflakes [6]–[8]. Furthermore, smoothed-particle hydrodynamics is used as a modified Lagrangian method [9], and vortex filaments [10], [11] are widely used to portray volumetric flows. Research efforts have been further extended to methods of preserving kinetic energy [12], [13], accelerating pressure projection [14], [15], and increasing advection accuracy [16]–[18]. In particular, VFX-related studies [19]–[21] have been acknowledged by artists for their ease of use, efficiency, and desirable results.

Accordingly, our study focused on increasing artists' efficiency by reducing the computational cost while maintaining the quality of the existing method. This would allow artists to quickly view the results of their designed scenes.

B. FIRE SIMULATION

Research regarding realistic fire simulation methods commenced in the mid-1990s [22]–[25]. Interactive fire simulations [24] have been created based on grid-based fluid simulations [5]. Level-based fire simulation studies [26] have successfully depicted realistic fire motion by modeling the velocity and pressure at the interface of the fuel and burned products. Subsequently, the detonation shock dynamics technique [27] was proposed to depict flame-wrinkled patterns. In addition to the depiction of fire, techniques useful to artists have been investigated, enabling the modeling of a burning rigid body and surface with volumetric grids [28]–[30].

These studies have led to high-quality fire simulations; however, the simulations require a significant amount of time. Hence, a multi-GPU framework for parallelization of the particle simulation was proposed to accelerate the existing hybrid simulation [31]. Since then, the focus of fire simulation research has shifted from the depiction of fire to increasing artists' efficiency. A temperature control method to simulate flames in the form of target shapes was proposed [32]. Thereafter, a method to generate fire motion and natural fire flakes as a secondary effect of fire simulation was proposed [1]. Recently, a method to analyze actual input videos and animated fire videos was proposed to create physical fields and fire flakes that appear as natural as those from the input videos [33].

However, none of the aforementioned methods operate in real time. They require a significant amount of time to obtain and modify the results. Hence, in this study, we adopted a neural network model to obtain the desired results more quickly and to address the challenges of non-real-time methods.

C. MACHINE LEARNING-BASED MODEL FOR FLUID SIMULATION

Fluid simulation involves multiple steps, including data generation, equation solving, and rendering to create the final outputs. In particular, the equation solver is important for solving fluid mechanics with a range of approaches, such as Lagrangian and Eulerian approaches based on the Navier–Stokes equation for advection and projection.

Researchers have used machine learning to facilitate fluid simulation. However, as several steps must be performed to obtain the final outputs, machine learning may not completely replace the existing method. Some researchers have proposed methods to partially replace each simulation step with machine learning. In grid-based fluid simulation, the projection step is replaced with a neural network [34] to successfully accelerate the computation compared with the traditional method. Furthermore, in Eulerian fluid simulation, the pressure is determined using unsupervised learning and a convolution neural network (CNN) [35], in which stable results are obtained more quickly. A long short-term memory-based method [36] was proposed to predict the pressure change in multiple subsequent time steps and significantly improve the speed of the pressure solver.

Following the research on partially applying machine learning to the simulation steps, a method was proposed that used regression forests for approximation rather than numerically solving the Navier–Stokes equation [37]. This approximation method in the Lagrangian system successfully inferred the velocity and position of particles in the next time step. Similarly, a neural network has been used [38] to approximate the fluid particle behavior or nonlinear dynamics of the existing method. A conditional generative adversarial network-based method [39], [40] was proposed to solve two-dimensional advection–diffusion problems. Furthermore, studies have been conducted on the space-time deformation of interactive liquids [41]. A method using CNN-based descriptors was proposed to obtain precomputed patches [42]. TempoGAN [43] was proposed as a high-quality high-resolution simulation method that

FIGURE 1. Generating fire-flake animation natural to flames and surroundings in appropriate locations with neural-network-based fire-flake simulation model.

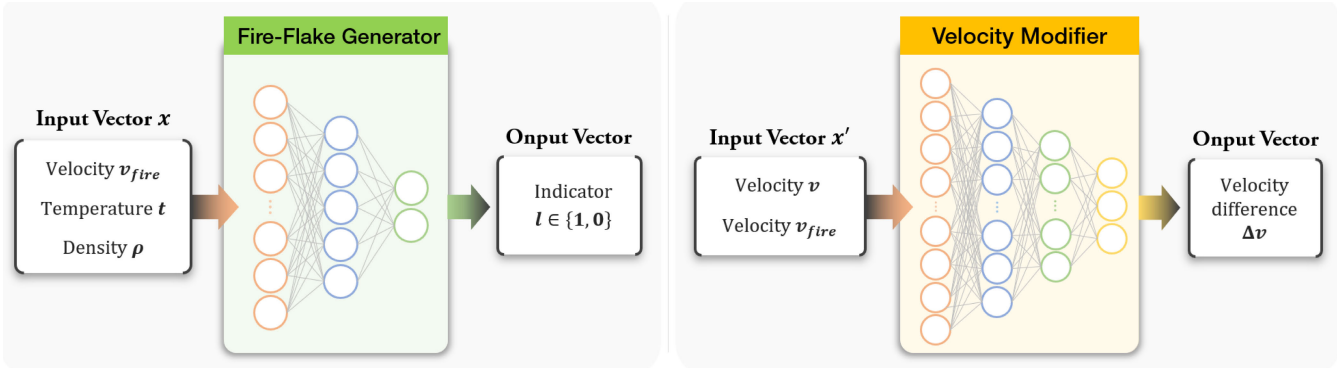

FIGURE 2. Structures of our neural networks.

synthesizes four-dimensional physics fields using neural networks. A novel generative model was also proposed to synthesize fluid simulations from a set of reduced parameters [44]. The first transport-based neural style transfer algorithm for volumetric smoke data was proposed to transfer features from natural images to smoke simulations [45]. Subsequently, an end-to-end learning approach for the overall simulation process was proposed [46]. Furthermore, various features of the Lagrangian fluid simulation data were trained via CNN.

III. NEURAL NETWORK-BASED FIRE-FLAKE MODEL

The proposed neural network-based fire-flake model comprises two neural network components and an integration step [47]. The neural network components are a fire-flake generator and a fire-flake velocity modifier. The structures of the neural networks are illustrated in Fig. 2. The fire-flake generator analyzes the input temperature, density, and velocity fields to determine the appropriate locations for fire flakes to be generated. Once the fire flakes are generated, the velocity modifier analyzes the input velocity field and temperature field to infer the velocity of each fire flake to naturally suit the flames and surroundings.

The integration step uses the Euler method to integrate the newly inferred fire-flake velocity and determine the next position. Next, each fire flake either moves to the calculated position or is removed if it crosses the simulation boundary or if its velocity decreases below a user-defined threshold th_{random}. These operations repeat in each frame.

Section 3.1 describes the structure of the fire-flake generator. Section 3.2 describes the structure of the fire-flake velocity modifier. Section 3.3 presents the data required to train the two neural network components. Section 3.4 describes the optimization to control the intuitive fire-flake generation and improve its performance.

A. FIRE-FLAKE GENERATOR

The fire-flake generator $\mathbf{G}$ performs neural network-based classification to determine the locations at which fire flakes will be generated from precomputed fire simulation data. The training data for the generator contain, for each grid position

p, the indicator value, which indicates whether the fire flake was generated $l \in \{1, 0\}$, and the input feature vector **x**, comprising the fire velocity $\mathbf{v}_{fire}$, temperature t, and density ρ. The generator **G** is trained on the input feature vector $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ and its target values (indicator l). After training, the fire-flake generator receives the input vector **X** to infer the probability of fire-flake generation P_s in each simulation grid position **p** as follows:

$$P_s(l_i|\mathbf{x}_i) \sim P(l_i|\mathbf{y}_s(\mathbf{x}_i|\mathbf{w}_s)) \tag{1}$$

$\mathbf{y}_s(\mathbf{x}_i|\mathbf{w}_s)$ refers to the target probability distribution denoted by the weight $\mathbf{w}_s$ and i is the grid index. Ultimately, this can be deduced from the likelihood function in line with $\mathrm{L} = \{l_1, l_2, \cdots, l_N\}$ as follows:

$$L(\mathbf{L}|\mathbf{X}) = \prod_{i=1}^{N} P(l_i|\mathbf{y}_s(\mathbf{x}_i|\mathbf{w}_s)) \tag{2}$$

To maximize the likelihood, we determine the probability of fire-flake generation using Softmax. The fire-flake generator **G** outputs the probability vector pertaining to the generation or non-generation of flakes from the feature vector **x** in the grid position **p**. Subsequently, the loss is calculated using the binary cross-entropy function based on the generator outputs and the target value (indicator value l).

Thus, after being trained on the training data for fire-flake generation distribution, the generator **G** infers the generation of fire-flakes based on the generation probabilities in specific grid positions.

B. VELOCITY MODIFIER

The velocity of a fire flake can be modeled as a summation of its external forces. According to the existing method [32], the external forces are the fire velocity, lift force, vorticity, and buoyancy, which can be represented as follows:

$$\mathrm{F}_{total} = \mathrm{F}_{flow} + \mathrm{F}_{drag} + \mathrm{F}_{lift} + \mathrm{F}_{bouy} \tag{3}$$

$$\mathrm{F}(\mathrm{v}, t)_{total} = \mathrm{c}_{flow}\mathrm{v} + \mathrm{c}_{drag}|\mathrm{v}'|^2\mathrm{v}'_{dir} + \mathrm{c}_{lift}t\mathrm{v} \times \omega_{flow} + c_{bouy}t\mathrm{v}_{up}, \tag{4}$$

where c_{flow} is the flow coefficient, c_{drag} is the drag force coefficient, c_{lift} is the lift force coefficient, and c_{bouy} is the buoyancy coefficient. v is the velocity of the fire flake, t is the temperature of the fire flake, and $|\mathrm{v}'|$ is the relative velocity between the fire-flake velocity and v_{flow}. v_{flow} is the velocity of the fire simulation and $\omega_{flow} = \nabla \times v_{flow}$, whereas v_{up} is the upward normal vector. Therefore, F_{total} can be represented as a vector function. Hence, unlike the previous work [47] based on a normal distribution, our neural network is based on polynomial regression.

To calculate F_{total} using Equation (4) in grid position **p**, the physical quantities of the grid position **p** and its neighbors are required. Hence, the fire velocity values of the grid position **p** and its six neighbors were used for training (see Fig. 3).

The proposed velocity modifier **M** does not instantly infer the fire-flake velocity in the current frame. Instead, it infers

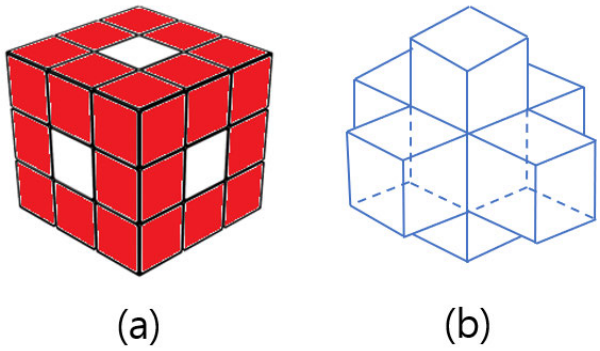

FIGURE 3. When calculating F_{total}, we use the velocity data of seven grid positions, as shown by deleting all edge position cubes (red) in (a). The results appears as in (b).

the change $\Delta\mathbf{v}$ in fire-flake velocity between the preceding and current frames. Hence, the training data contain the per-particle velocity differences between the fire flakes $\Delta\mathbf{v}_{flake} = \mathbf{v}^{t+1}_{flake} - \mathbf{v}^{t}_{flake}$ for the target value and the input feature vector $\mathbf{x}'$ consisting of the per-grid fire velocities $\mathbf{V}^{t}_{fire} = \{\mathbf{v}^{t}_{fire,\mathbf{p}1}, \mathbf{v}^{t}_{fire,\mathbf{p}2}, \cdots, \mathbf{v}^{t}_{fire,\mathbf{p}7}\}$ and per-particle fire-flake velocities $\mathbf{v}^{t}_{flake}$ at time step t. The modifier **M** is trained with the input feature vector $\mathbf{X}' = \{\mathbf{x}'_1, \mathbf{x}'_2, \cdots, \mathbf{x}'_N\}$ and its target values ($\Delta\mathbf{v}_{flake}$). Thereafter, once the feature vector $\mathbf{X}'$ enters the velocity modifier as an input, the output is not the absolute value of the advected velocity but the change in velocity, $\Delta\mathbf{V} = \{\Delta\mathbf{v}_1, \Delta\mathbf{v}_2, \cdots, \Delta\mathbf{v}_N\}$. As the data inputs may have different velocity distributions, inferring the change instead of the absolute value allows the velocity modifier to be applied more generally.

Subsequently, to minimize the gap between the inferred velocity difference and the target value, the following least-square error is used:

$$Error = \sum_{i=1}^{N} |\Delta\mathbf{v}_i - M(x'_i)|^2 \tag{5}$$

$M(x'_i)$ is the inferred velocity difference, and $|\Delta\mathbf{v}_i|$ is the target value. Subsequently, the inferred value is used to update the velocity and position of a flake as follows:

$$\mathbf{v}^{t+1}_{flake} = \mathbf{v}^{t}_{flake} + \Delta\mathbf{v}^{t} \tag{6}$$

$$\mathbf{p}^{t+1} = \mathbf{p}^{t} + \mathbf{v}^{t+1}_{flake} \cdot \Delta t \tag{7}$$

where $\Delta\mathbf{v}^t$ is the velocity difference from modifier **M**. If a fire flake is positioned outside the simulation domain in this step, it is eliminated.

C. TRAINING DATA

We arranged the training data based on the traditional fire and fire-flake simulation framework [1]. The training data included six fire scenes in which the mesh-shaped fuel was set alight and some meshes were in motion, as shown in Fig. 4. All scenes, including the boundaries, had a grid resolution of $65 \times 130 \times 130$. Furthermore, 300 frames were used for the

attack and down scenes, whereas 100 frames were used for the other scenes (see Table 1).

As inputs to the fire-flake generator, the velocity, temperature, and density fields of the precomputed fire simulation data were inserted into the feature vector. Therefore, the input feature vector had five components. The target values for the generation or non-generation of fire flakes formed the indicator value l. We set the target values to 1 for fire flakes generated from the traditional fire-flake simulation across all frames and grids, and we set the target value to 0 when no fire flakes were generated. We saved the grid index values and binary data. We also generated binary data for all frames in each simulation and saved the feature vector data for the corresponding grid indices. Next, we performed 1:1 sampling using generated and non-generated fire-flake data, adjusted the ratio equally, and finalized the training data.

Unlike the fire-flake generator, the only input for the velocity modifier was the velocity value. The input feature vector consisted of the flake velocities when flakes were present in the grid and the fire velocities for the grid. The fire velocities included the velocity of the reference grid and those of the six neighboring grids3. Therefore, the input feature vector had 24 components. The fire-flake velocity change between the preceding and current frames before and after their advection was used as the target. In addition to the advection of the initially generated flakes, we backtracked continuous advection steps between frames to maximize the training data for the velocity modifier. Subsequently, by analyzing the velocity distribution across the six scenes, we performed sampling to place each distribution in the training data as regularly as possible and finalized the training dataset. In developing the training data, the 1:1 sampling and velocity difference returned encouraging results.

As different forms of feature vectors were used for the fire-flake generator and velocity modifier, we collected and saved the relevant data separately. There were 2,507,070 fire-flake samples and 1,757,892 velocity modifier samples in the training data, of which 501,411 and 351,578 were used for the validation datasets, respectively.

D. OPTIMIZATION

In the traditional fire-flake simulation [1], there are three user parameters to control fire-flake generation. The parameters consist of random variables, the difference of fire energy and temperature for controlling the number of fire flakes to be generated. The user can set these variables manually to obtain different results. However, it is difficult to predict the results that result from a given combination of variables. Our method addresses this issue by combining the three parameters into a single parameter. Accordingly, the training data were generated from a scene with a large number of fire flakes, and a random variable r was assigned to a new fire-flake. After the generator $\mathbf{G}$ is trained with this data, it generates new fire flakes and we eliminate the fire flakes that fail to satisfy the user control parameter th_{random} as follows:

$$\text{fire-flake} = \begin{cases} \text{not eliminated}, & \text{if } r \leq th_{random} \\ \text{eliminated}, & \text{if } r > th_{random} \end{cases} \quad (8)$$

In this step, although we could enable the fire-flake generator to view the entire grid in our model, we only allow it to determine whether to generate fire flakes in the grids where the fire density is non-zero. This optimization method provides faster overall performance.

IV. RESULTS

A. IMPLEMENTATION DETAILS

In this study, all processes were executed using an Intel i7-8700 3.20 GHz CPU, NVIDIA GeForce GTX 1080 Ti GPU, and 32 GB RAM. The animated fire-flake results were rendered using Autodesk Maya 2016. We used the TensorFlow deep-learning framework to build and train the fire-flake generator and velocity modifier.

The details of each network structure are as follows. The fire-flake generator contained three fully connected layers, whereas the velocity modifier contained four. Each layer L contained the learnable parameters weight w_L and bias b_L. The layers were connected to the activation function σ and can be represented as $y_L = \sigma(w_L y_{L-1} + b_L)$. The three layers in the fire-flake generator contained 15, 5, and 2 nodes, whereas the four layers in the velocity modifier contained 30, 24, 12, and 3 nodes. Both networks used ReLU as the activation function with a weight decay applied and ADAM as the optimizer. The two networks were trained independently. The learning rate of the generator was 10^{-5} and that of the modifier was 10^{-4}. Each was provided with 50,000 batches. Furthermore, 10,000 and 300,000 training steps were used for the generator and modifier, respectively.

B. LOCATIONS OF FIRE-FLAKES GENERATED IN FLAME

In a fire-flake simulation, fire flakes must be generated in appropriate locations. Fire flakes are small particles created during combustion, caused by internal and external impacts, and they are generated both on the surface of and inside the burning materials. Fig. 4 shows fire flakes generated in six fire simulations with different inputs. As can be seen, all fire flakes were generated appropriately on the surface and inside the fire. In particular, excluding scenes #3 and #4 in Fig. 4, all other scenes show that the fire location varied in each frame because the fire was in motion. Fig. 1 presents the results of scene #6 in Fig. 4 at an interval of 30 frames. Our model generated fire flakes in appropriate locations without errors despite the fire being in motion.

C. MOTION OF FIRE-FLAKES SYNCHRONIZED WITH FLAMES

Once fire flakes are generated in accurate locations, the next step is to generate harmonious motion of the fire flakes in the velocity field. In Fig. 4, the fire flakes are moving upward and are affected primarily by buoyancy in the absence

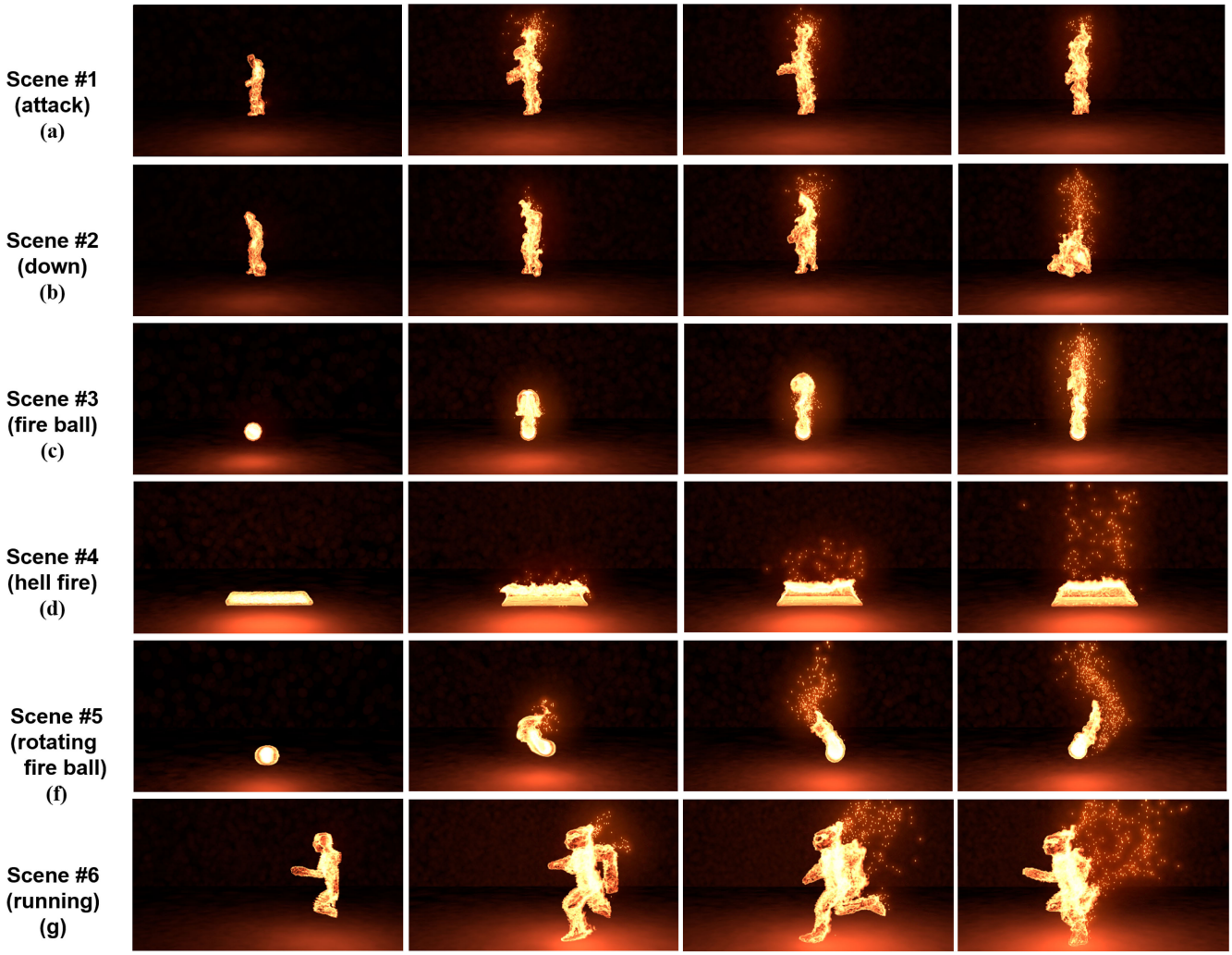

FIGURE 4. Scenes #1 to #6, which are called attack, down, fireball, hellfire, rotating fireball, and running, respectively. The images show the fire-flake simulation results from our model in time lapse.

of any external force. In contrast, in Fig. 5, the fire flakes exhibit natural motion, which was inferred from the external force. Fig. 5(c) shows the velocity field corresponding to the images in the top rows. The proposed neural network model inferred the natural motion from the external force from the surroundings, although the external force was not considered in the training.

As discussed in the Results section, the fire flakes generated in our model were visually natural. In this section, we quantitatively evaluate whether our model is well-trained.

1) FIRE-FLAKE GENERATOR

The fire-flake generator received fire data inputs and returned locations of the fire flakes to be generated in each frame. To evaluate whether the number of newly generated fire flakes was appropriate, we compared the number of new fire flakes generated using our model with those generated using the existing method, as shown in Fig. 6. To adjust the results for a fair comparison, we set the value of th_{random} so that the numbers of flakes in the final frames were comparable. The numbers of flakes generated in the running, fireball, and attack scenes were comparable to those generated using the existing method. Similarly comparable trends were obtained for the other scenes.

Fig. 7 illustrates the total number of fire flakes generated in the scene and rendered in each frame. Because the number of fire flakes generated in a scene and visualized in each frame were comparable between our model and the existing method, it can be concluded that our model successfully approximated the trend of the training data. We measured the numbers of flakes across the six scenes and set th_{random} such that the numbers of flakes in the final frames were comparable. We found that our model performed well as it yielded results that were comparable to those in the synchronized fire-flake simulation.

In the hellfire, fireball, and rotating fireball scenes, the numbers of flakes generated by the existing solver in the

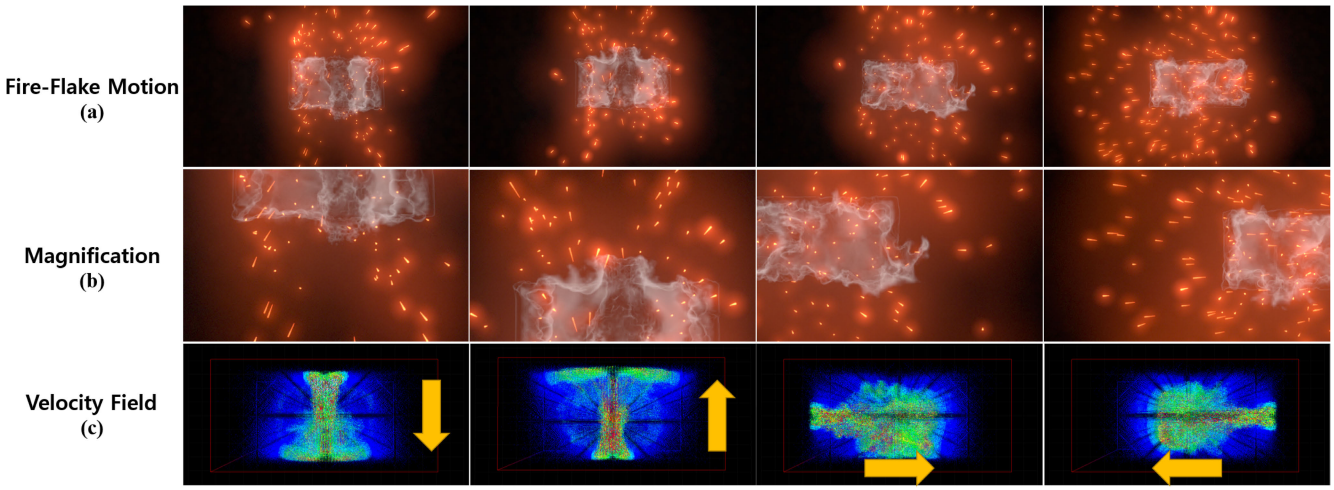

FIGURE 5. Illustration of fire flakes in motion affected by external force, a magnified view, and the velocity field.

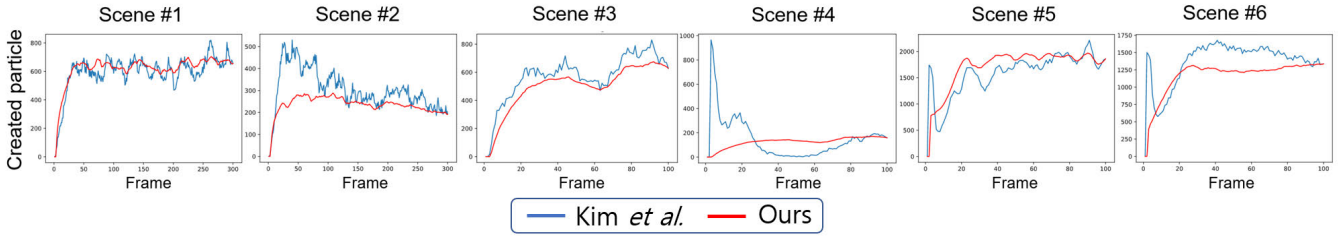

FIGURE 6. Number of flakes newly generated in each frame with Kim *et al.* vs. our method.

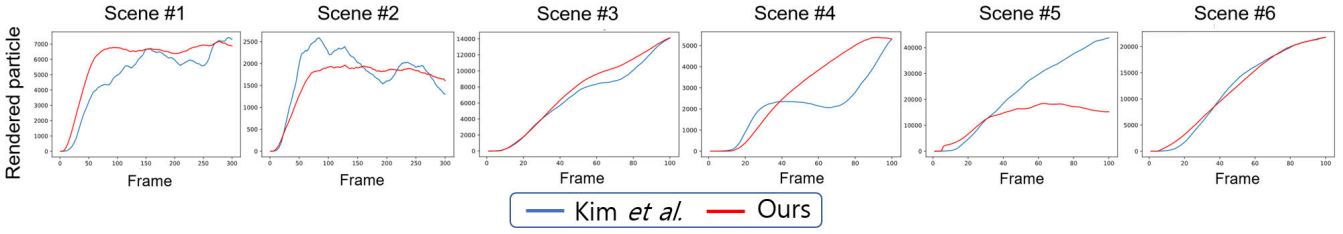

FIGURE 7. Number of flakes to be rendered in each frame with Kim *et al.* vs. our method.

initial frames were greater than those in other frames. This can be attributed to the fire simulation when the fuel source's velocity abruptly changes from the initial velocity of 0 to a non-zero velocity in the next frame. Such sections were corrected in the advection step of the existing solver. Nonetheless, in our model, the number of flakes increased seamlessly following the first frame, and the generator successfully generated the initial flakes without any correction in the advection step.

2) VELOCITY MODIFIER

The velocity modifier infers the natural velocity of the previously and newly generated fire flakes in each frame. Subsequently, the integration step proceeds with advection from the first to last frames. Hence, the velocity subjected to advection in the preceding frame affects the fire-flake position in the following frame. If errors in all the frames accumulate, the obtained result in the final frame is highly likely to deviate from the desired result. Thus, the velocity modifier is crucial across the entire animation. A substantial gap between the velocity of the fire in motion and that of the fire flakes makes the scene appear unrealistic. Hence, the velocity of the fire flakes must be comparable to the velocity of the fire to create a natural scene.

For the evaluation, we calculated the average magnitude of the velocity of the fire flakes to be rendered in each frame and compared the results with those generated using the existing method. If the magnitude of the fire-flake velocity changed naturally across all frames, we set th_{random} to a value comparable to the result from the existing method per scene instead of using a fixed value. As shown in Fig. 8, in all scenes, the magnitude of the fire-flake velocity changed similarly to

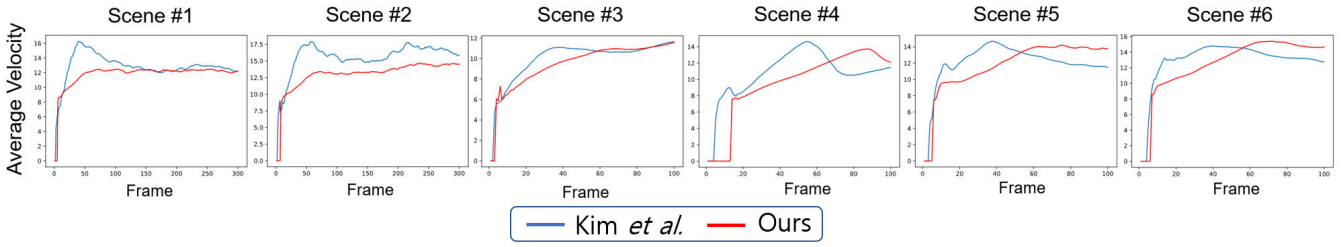

FIGURE 8. Average velocity of flakes to be rendered in each frame with Kim *et al.* vs. our method.

TABLE 1. Training data statistics and inference performance of synchronized fire-flake simulation vs. our method.

#	Scene Name	# of frames	Total # of fire-flakes		Avg. Born Time (s) / Frame		Avg. Advection Time (s) / Frame		Avg. Run Time (s) / Frame		Total Processing Time (s) / Scene	
			[Kim *et al.*]	Ours	[Kim *et al.*]	Ours	[Kim *et al.*]	Ours	[Kim *et al.*]	Ours	[Kim *et al.*]	Ours
1	Scene #1	300	1,501,347	1,501,711	0.039	0.033	0.044	0.001	0.084	0.035	25.248	10.611
2	Scene #2	300	531,166	531,288	0.035	0.033	0.038	0.001	0.074	0.034	22.296	10.455
3	Scene #3	100	2,259,126	2,259,207	0.038	0.032	0.047	0.004	0.085	0.036	8.551	3.688
4	Scene #4	100	215,791	215,850	0.038	0.029	0.035	0.001	0.073	0.030	7.340	3.026
5	Scene #5	100	1,182,791	1,182,998	0.033	0.027	0.041	0.002	0.075	0.029	7.531	2.998
6	Scene #6	100	663,155	663,166	0.038	0.029	0.053	0.001	0.091	0.031	9.160	3.142

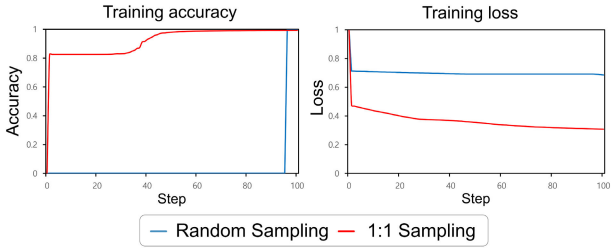

FIGURE 9. Accuracy and loss comparison among sampling methods used in the fire-flake generator.

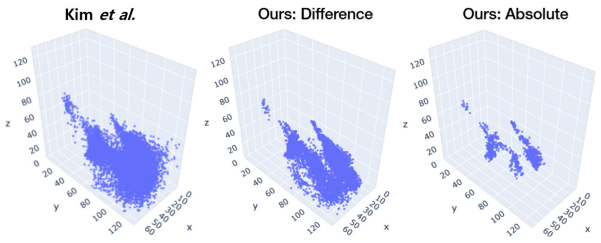

FIGURE 10. Locations for generating fire-flakes in velocity modifier in line with ground-truth settings. Ours: Difference represents the result when the ground truth of the velocity modifier is the velocity difference, and Ours: Absolute represents the result when the ground truth is the absolute velocity value.

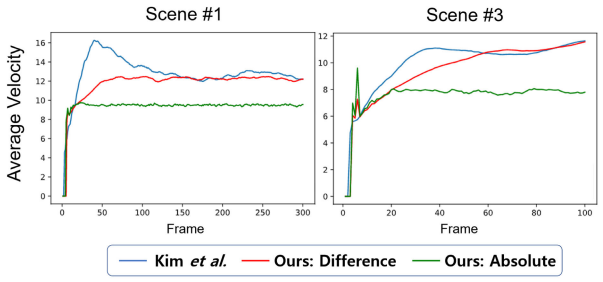

FIGURE 11. Average velocity in velocity modifier in line with ground-truth settings.

the average magnitude of the velocity in the existing solver, which validates our method's velocity inference.

3) PERFORMANCE

Table 1 shows a comparison of our model's performance metrics with those of the existing fire-flake simulation. The average time required to determine the locations at which to generate fire flakes and to update the flake generation information is denoted by Avg. Born in the table. In our model, Avg. Born represents the performance of the fire-flake generator and the th_{random} used for adjusting the fire-flake count and updating the generation information.

Avg. Advection is the average time required to update the changes in velocity and position of the fire flakes affected by external forces. In our model, Avg. Advection represents the performance of the velocity modifier and the velocity and position updates of the fire flakes. The Avg Advection values show our method was approximately 24 times faster on average than the existing method. In previous studies, the drag force, vorticity, and other external forces had to be numerically calculated to obtain the advected velocity and position; however, our method uses the velocity modifier network to calculate them simultaneously.

In the previous study, simulation results were generated for 12 frames per second on average, whereas our model generated approximately 30 frames per second on average. This means that our model is an average of 2.4 times faster than the previous model and enables near-real-time simulations.

V. DISCUSSION AND LIMITATIONS

In this study, we found that the composition of training data of the fire-flake generator affects the results. In all frames, the fire-flake indicator value l is far more often 0 rather than 1. This could be explained by the fact that the area the fire

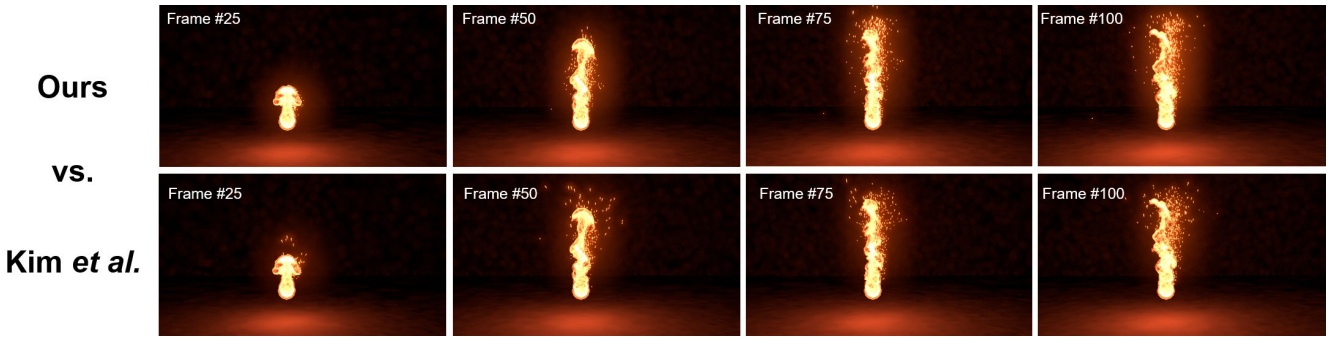

FIGURE 12. The first row is the fire flakes generated using our method, whereas the second is the fire flakes generated using the method of Kim *et al.*

occupies is narrower than the simulation space. In this case, when l is randomly sampled, the area where fire flakes are not generated dominates the training data. Furthermore, the area where fire flakes are not generated is likely to have no or extremely low velocity, temperature, and density and consist of overlapping or similar data. This complicates training the generator under accurate conditions for generating fire flakes. We created the training data by adjusting the areas where fire flakes were or were not generated with 1:1 sampling instead of a simple random sampling of l. Fig. 9 shows the training accuracy and training loss of the sampling methods (i.e., random sampling vs. 1:1 sampling) used to create the training data for the generator. As shown in Fig. 9, the 1:1 sampling was more accurate and performed fire-flake generation more appropriately than random sampling. In the case of random sampling, the accuracy increased rapidly near the last step, which means that it was considerably overfitted to data where a large number of fire flakes were not generated; therefore, nearly all labels were output as zero.

In the velocity modifier, we found that the label composition of the training data affects the training results. Fig. 10 shows the locations where fire flakes were generated in line with the ground-truth settings. When the velocity difference is set to the ground truth, each fire flake is generated at an appropriate location, and the number of fire flakes is similar to that in the previous study. Fig. 11 demonstrates the average velocity used in the comparison. When the absolute velocity value is the ground truth, the average velocity does not change over time and is approximately constant. These results indicate that rather than learning by setting the absolute velocity value as the ground truth for training the velocity modifier, learning by setting the velocity difference between the preceding and current frames as the ground truth generates a more natural fire-flake velocity. This is because inferring the quantitative change based on the velocity of the preceding frame is less prone to errors than inferring a new fire-flake velocity.

Fig. 12 shows that our results and those of the Kim *et al.* model have a similar distribution of generating fire flakes. The results are not exactly the same; our model was faster than those in previous studies and maintained visual fidelity. However, it has limitations hindering its comparison with physically strict simulations. This is because fire flakes are small carbon-based materials that are not completely free from interactions with the surroundings. This limitation should be addressed for a more accurate simulation. A fire-flake model that interacts with the surroundings can be implemented if additional methods are employed to calculate the effects of the surrounding velocity fields on fire flakes when creating the training data.

VI. CONCLUSION

This is the first study using neural networks for fire-flake simulation that adopts neural networks to replace most of the operations necessary in the existing method. The fire-flake generator determines the appropriate locations for generating fire flakes. The velocity modifier infers the fire-flake velocity based on polynomial regression to make them fit naturally in their surroundings. Both steps maintain the visual plausibility of the existing simulation while improving the performance. Furthermore, we improved on existing fire-flake simulation by combining multiple variables into a single intuitive variable. These results will improve the efficiency of artists. In future work, we will adopt deep learning in fire simulations to obtain faster simulation results. A pipeline will be developed to integrate deep learning with our neural network-based fire-flake model, which will accelerate the entire process, thereby ensuring more natural results.

REFERENCES

[1] T. Kim, E. Hong, J. Im, D. Yang, Y. Kim, and C.-H. Kim, "Visual simulation of fire-flakes synchronized with flame," *Vis. Comput.*, vol. 33, nos. 6–8, pp. 1029–1038, Jun. 2017. [Online]. Available: https://link.springer.com/article/10.1007/s00371-017-1374-9
[2] C. Cummins, M. Seale, A. Macente, D. Certini, E. Mastropaolo, I. M. Viola, and N. Nakayama, "A separated vortex ring underlies the flight of the dandelion," *Nature*, vol. 562, no. 7727, pp. 414–418, Oct. 2018.
[3] R. Bridson, *Fluid Simulation for Computer Graphics*. Boca Raton, FL, USA: CRC Press, 2015.
[4] F. H. Harlow and J. E. Welch, "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *The Phys. Fluids*, vol. 8, no. 12, pp. 2182–2189, 1965.
[5] J. Stam, "Stable fluids," *Proc. 26th Annu. Conf. Comput. Graph. Interact. Techn.*, 1999, pp. 121–128, doi: 10.1145/311535.311548.
[6] C. Jiang, C. Schroeder, J. Teran, A. Stomakhin, and A. Selle, "The material point method for simulating continuum materials," in *Proc. ACM SIGGRAPH Courses*, Jul. 2016, pp. 1–52.

[7] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, "A material point method for snow simulation," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–10, Jul. 2013.
[8] A. P. Tampubolon, T. Gast, G. Klár, C. Fu, J. Teran, C. Jiang, and K. Museth, "Multi-species simulation of porous sand and water mixtures," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–11, Jul. 2017.
[9] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "SPH Fluids in Computer Graphics," in *Eurographics*, S. Lefebvre and M. Spagnuolo, Eds. Genoa, Italy: The Eurographics Association, 2014.
[10] A. Angelidis, F. Neyret, K. Singh, and D. Nowrouzezahrai, "A controllable, fast and stable basis for vortex based smoke simulation," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*. Genoa, Italy: Eurographics Association, 2006, pp. 25–32.
[11] S. Weißmann and U. Pinkall, "Filament-based smoke with vortex shedding and variational reconnection," in *Proc. ACM SIGGRAPH Papers*, 2010, pp. 1–12.
[12] R. Fedkiw, J. Stam, and H. W. Jensen, "Visual simulation of smoke," in *Proc. 28th Annu. Conf. Comput. Graph. Interact. Techn.*, 2001, pp. 15–22.
[13] A. Selle, N. Rasmussen, and R. Fedkiw, "A vortex particle method for smoke, water and explosions," in *Proc. ACM SIGGRAPH Papers*, 2005, pp. 910–914.
[14] A. McAdams, E. Sifakis, and J. Teran, "A parallel multigrid Poisson solver for fluids simulation on large grids," in *Proc. 2010 ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*. Genoa, Italy: Eurographics Association, 2010, pp. 65–74.
[15] R. Setaluri, M. Aanjaneya, S. Bauer, and E. Sifakis, "SPGrid: A sparse paged grid structure applied to adaptive smoke simulation," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 1–12, Nov. 2014.
[16] B. Kim, Y. Liu, I. Llamas, and J. R. Rossignac, "Flowfixer: Using BFECC for fluid simulation," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep. GIT-GVU-05-24, 2005.
[17] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac, "An unconditionally stable MacCormack method," *J. Sci. Comput.*, vol. 35, nos. 2–3, pp. 350–371, Jun. 2008.
[18] J. Zehnder, R. Narain, and B. Thomaszewski, "An advection-reflection solver for detail-preserving fluid simulation," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–8, Aug. 2018.
[19] M. B. Nielsen, B. B. Christensen, N. B. Zafar, D. Roble, and K. Museth, "Guiding of smoke animations through variational coupling of simulations at different resolutions," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*, 2009, pp. 217–226.
[20] Z. Pan and D. Manocha, "Efficient solver for spacetime control of smoke," *ACM Trans. Graph.*, vol. 36, no. 4, p. 1, Jul. 2017.
[21] L. Shi and Y. Yu, "Taming liquids for rapidly changing targets," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*, 2005, pp. 229–236.
[22] J. Stam and E. Fiume, "Depicting fire and other gaseous phenomena using diffusion processes," in *Proc. 22nd Annu. Conf. Comput. Graph. Interact. Techn.*, New York, NY, USA, 1995, pp. 129–136, doi: 10.1145/218380.218430.
[23] P. Beaudoin, S. Paquet, and P. Poulin, "Realistic and controllable fire simulation," in *Proc. Graph. Interface*, 2001, pp. 159–166. [Online]. Available: https://dl.acm.org/doi/10.5555/780986.781006, doi: 10.5555/780986.781006.
[24] Z. Melek and J. Keyser, "Interactive simulation of fire," in *Proc. 10th Pacific Conf. Comput. Graph. Appl.*, 2002, pp. 431–432. [Online]. Available: https://ieeexplore.ieee.org/document/1167889
[25] A. Lamorlette and N. Foster, "Structural modeling of flames for a production environment," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 729–735, Jul. 2002, doi: 10.1145/566654.566644.
[26] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen, "Physically based modeling and animation of fire," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 721–728, Jul. 2002, doi: 10.1145/566654.566643.
[27] J.-M. Hong, T. Shinar, and R. Fedkiw, "Wrinkled flames and cellular patterns," *ACM Trans. Graph.*, vol. 26, no. 3, p. 47, Jul. 2007, doi: 10.1145/1276377.1276436.
[28] S. Liu, T. An, Z. Gong, and I. Hagiwara, "Physically based simulation of solid objects' burning," in *Transactions on Edutainment VII*. New York, NY, USA: Springer, 2012, pp. 110–120.
[29] Y. Zhao, X. Wei, Z. Fan, A. Kaufman, and H. Qin, "Voxels on fire [computer animation]," in *Proc. IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, 2003, pp. 271–278.
[30] N. Chiba, K. Muraoka, H. Takahashi, and M. Miura, "Two-dimensional visual simulation of flames, smoke and the spread of fire," *J. Vis. Comput. Animation*, vol. 5, no. 1, pp. 37–53, Jan. 1994. [Online]. Available: https://link.springer.com/article/10.1007/s00371-016-1267-3
[31] C. Horvath and W. Geiger, "Directable, high-resolution simulation of fire on the GPU," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–8, Jul. 2009, doi: 10.1145/1531326.1531347.
[32] T. Kim, J. Lee, and C.-H. Kim, "Physics-inspired controllable flame animation," *Vis. Comput.*, vol. 32, nos. 6–8, pp. 871–880, Jun. 2016. [Online]. Available: https://link.springer.com/article/10.1007/s00371-016-1267-3
[33] J.-H. Kim and J. Lee, "Fire sprite animation using fire-flake texture and artificial motion blur," *IEEE Access*, vol. 7, pp. 110002–110011, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8793066
[34] C. Yang, X. Yang, and X. Xiao, "Data-driven projection method in fluid simulation," *Comput. Animation Virtual Worlds*, vol. 27, nos. 3–4, pp. 415–424, May 2016.
[35] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating eulerian fluid simulation with convolutional networks," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 3424–3433.
[36] S. Wiewel, M. Becher, and N. Thuerey, "Latent space physics: Towards learning the temporal evolution of fluid flow," in *Computer Graphics Forum*, vol. 38, no. 2. Hoboken, NJ, USA: Wiley, 2019, pp. 71–82.
[37] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven fluid simulations using regression forests," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 1–9, Nov. 2015.
[38] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *The J. Mach. Learn. Res.*, vol. 19, no. 1, pp. 932–955, 2018.
[39] A. Barati Farimani, J. Gomes, and V. S. Pande, "Deep learning the physics of transport phenomena," 2017, *arXiv:1709.02432*. [Online]. Available: http://arxiv.org/abs/1709.02432
[40] Z. Long, Y. Lu, X. Ma, and B. Dong, "PDE-net: Learning PDEs from data," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, J. Dy and A. Krause, Eds. Stockholm Sweden: PMLR, Jul. 2018, pp. 3208–3216. [Online]. Available: http://proceedings.mlr.press/v80/long18a.html
[41] L. Prantl, B. Bonev, and N. Thuerey, "Generating liquid simulations with deformation-aware neural networks," 2017, *arXiv:1704.07854*. [Online]. Available: http://arxiv.org/abs/1704.07854
[42] M. Chu and N. Thuerey, "Data-driven synthesis of smoke flows with CNN-based feature descriptors," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–14, Jul. 2017.
[43] Y. Xie, E. Franz, M. Chu, and N. Thuerey, "TempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–15, Aug. 2018.
[44] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," in *Computer Graphics Forum*, vol. 38, no. 2. Hoboken, NJ, USA: Wiley, 2019, pp. 59–70.
[45] B. Kim, V. C. Azevedo, M. Gross, and B. Solenthaler, "Transport-based neural style transfer for smoke simulations," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–11, Nov. 2019, doi: 10.1145/3355089.3356560.
[46] Y. Zhang, X. Ban, F. Du, and W. Di, "FluidsNet: End-to-end learning for lagrangian fluid simulation," *Expert Syst. Appl.*, vol. 152, Aug. 2020, Art. no. 113410.
[47] K. Um, X. Hu, and N. Thuerey, "Liquid splash modeling with neural networks," *Computer Graphics Forum*, vol. 37, no. 8, pp. 171–182, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/cgf.13522

MYUNGJIN CHOI received the B.A. degree from the Department of Computer and Radio Communications Engineering, Korea University, in 2013. His current research interests include fluid simulation and scientific visualization and deep learning.

JEONG A WI (Graduate Student Member, IEEE) received the B.S. degree in physics from Chung-Ang University, Seoul, South Korea, in 2019, where she is currently pursuing the M.S. degree in imaging engineering with the Graduate School of Advanced Imaging Science, Multimedia & Film. Her current research interests include deep learning and computer graphics.

TAEHYEONG KIM received the M.S. and Ph.D. degrees from Korea University, in 2011 and 2017, respectively. His current research interests include physically-based simulation, fluid animation, web-based real-time rendering, and deep learning.

YOUNGBIN KIM (Member, IEEE) received the B.S. and M.S. degrees in computer science and the Ph.D. degree in visual information processing from Korea University, in 2010, 2012, and 2017, respectively. From August 2017 to February 2018, he worked as a Principal Research Engineer with Linewalks. He is currently an Assistant Professor with the Graduate School of Advanced Imaging Science, Multimedia & Film, Chung-Ang University. His current research interests include data science and deep learning.

CHANG-HUN KIM is currently a Professor with the Department of Computer Science and Engineering, Korea University. His current research interests include fluid animation and mesh processing. He is also a member of the IEEE Computer Society and ACM.