Received June 25, 2021, accepted July 6, 2021, date of publication July 9, 2021, date of current version July 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3095904

Accelerated Smoke Simulation by Super-Resolution With Deep Learning on Downscaled and Binarized Space

BYEONG-SUN HONG[1], QIMENG ZHANG[1], CHANG-HUN KIM[1], JUNG LEE[2], AND JONG-HYUN KIM[3]

[1]Interdisciplinary Program in Visual Information Processing, Korea University, Seoul 02841, South Korea
[2]School of Software, Hallym University, Chuncheon 24252, South Korea
[3]School of Software Application, Kangnam University, Yongin 16979, South Korea

Corresponding author: Jong-Hyun Kim (jonghyunkim@kangnam.ac.kr)

This work was supported in part by the Hallym University Research Fund under Grant HRF-202011-009, in part by the Research and Development Program for Forest Science Technology provided by the Korea Forest Service (Korea Forestry Promotion Institute) under Project 2021390A00-2123-0105, and in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning, under Grant 2017R1C1B5074984. The work of Chang-Hun Kim was supported by the Basic Science Research Program through the NRF funded by the Ministry of Science, ICT and Future Planning, under Grant NRF-2019R1A2C1008244.

ABSTRACT In this paper, we propose a highly efficient method for synthesizing high-resolution(HR) smoke simulations based on deep learning. A major issue for physics-based HR fluid simulations is that they require large amounts of physical memory and long execution times. In recent years, this issue has been addressed by developing deep-learning-based super-resolution(SR) methods that convert low-resolution(LR) fluid simulation results to HR(High-resolution) versions. However, these methods were not very efficient because they performed operations even in areas with low density or no density. In this paper, we propose a method that can maximize its efficiency by introducing a downscaled and binarized adaptive octree. However, even if it is divided by octree, because the number of nodes increases when the resolution of the simulation space is large, we reduce the size of the space by multiscaling and at the same time perform binarization to preserve the density that may be lost in this process. The octree calculated in this process has a structure similar to that of a multigrid solver, and the octree calculated at coarse resolution is restored to its original size and used for HR expression. Finally, we apply the SR process only to those areas having significant density values. Using the proposed method, the SR process is significantly faster and the memory efficiency is improved. The performance of our method is compared with that of an existing SR method to demonstrate its efficiency.

INDEX TERMS Fluid simulations, deep learning, super-resolution, adaptive synthesizing, octree, physics-based simulations.

I. INTRODUCTION

Because of advances in deep learning in recent years, physics-based simulation fields such as character animations [13], [14], [29] and fluid simulations [2], [6], [15], [24] have also been remarkably improved because of deep learning. Accuracy and efficiency in fields such as style transfer [5], [16], [27], character motion control [17], [26], and numerical analysis [18] have also improved. However, in certain fields, it still suffers from the need for a large amount of computation.

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

In physics-based fluid simulation, it is an important issue that the amount of computation increases rapidly depending on the resolution, particularly for 3D simulation. To address this issue, there have been studies to express flow detail in HR simulation by adding a turbulence term to LR fluid simulation [7], [8], and recently, there have been attempts to apply deep-learning additionally. The existing deep learning methods for HR fluid simulation can be divided into two categories based on the type of application. The first category includes studies that try to solve simulation equations directly using deep-learning methods. [2], [11]. Studies in the second category enhance the details of input LR simulation data by transforming the data into turbulent styles [42] or upscaling

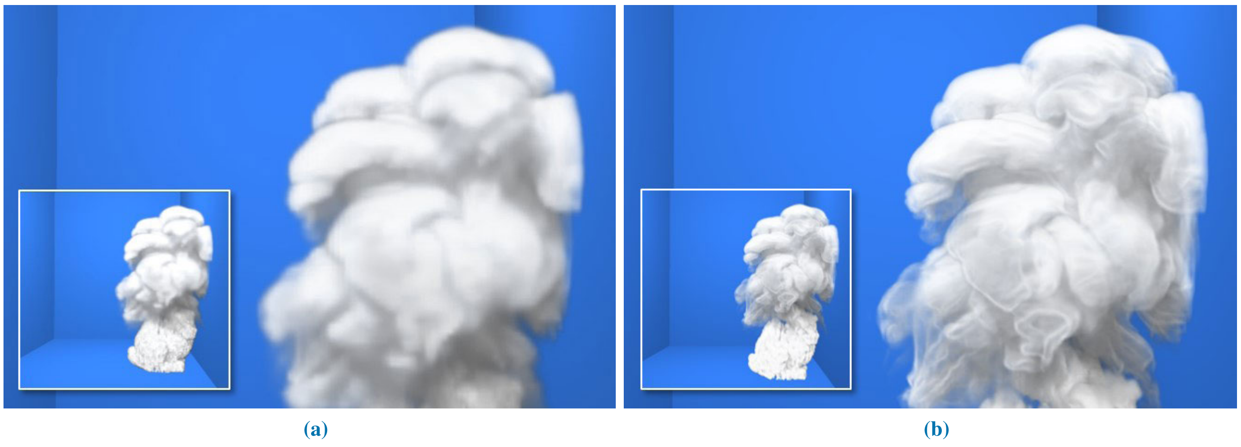


FIGURE 1. A super-resolution example of our method, which is based on tempoGAN [28]. (a) is the low-resolution input, (b) is the super-resolution result generated by our method.

its resolution [12], [28]. However, both the style-transform and resolution-upscaling methods are applied to the entire simulation space including the low-density region, reducing the efficiency of the computation process. In this paper, we propose a method that effectively addresses this issue by using a spatial partitioning method with an adaptive tree structure.

In this paper, we employ quadtree-based [34] and octree-based [35] structures to partition and store the simulation space. This enables the efficient production of HR results for ''smoke-like'' scenes. The SR method we used introduces the state-of-the-art work of tempoGAN [28]. Unlike the original tempoGAN, our method can reduce the amount of computation while maintaining quality by applying the operation only to the simulation area with significant density data. To make the SR result of Figure 1, our method takes 7.27 seconds and the original tempoGAN takes 90.03 seconds. According to the experimental results, as the density becomes more sparse, the performance improves. To compare the SR quality between our method and the original tempoGAN, we measured the difference error for each frame and confirmed that the difference was insignificant. Furthermore, because of the tree-based spatial partitioning technique, we can handle the SR operations of extremely high resolution without memory problems in GPUs.

In addition to the structural efficiency of octrees, we can significantly reduce computation time by applying binarization and downscaling techniques when constructing octrees. Our contributions are summarized as follows:

- We present the binarized and downscaled octree-based method to accelerate the SR of smoke scenes.
- Our method produces high-quality results while improving the performance of the SR process. Compared with existing methods, there is little quality degradation.

A. PROBLEM STATEMENT

In general, the grid-based spatial partitioning method, which is widely used in the Eulerian simulation technique, is mainly used to express smoke, water, and fire [19], [20]. Space can be optimized using data structures such as octree [21], [22], [25], but efficiency is not significantly improved because the amount of computation required for octree construction increases when the originally given grid resolution is high. A simple solution to this issue is to reduce the space to multiscale. However, in the process of downscaling the density, numerical dissipation occurs. As shown in Figure 2, numerical dissipation greatly occurs in the low-density regions (i.e., the region in which smoke is blurred, see the orange box in Figure 2), and this problem becomes more severe as downscaling proceeds.

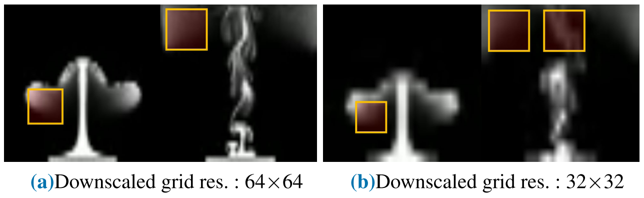

(a)Downscaled grid res. : 64×64
(b)Downscaled grid res. : 32×32

FIGURE 2. Example of numerical dissipation of density when performing downscaling (original grid res.: 128 × 128, orange box: regions with severe numerical dissipation).

In this paper, we propose a data optimization technique that can efficiently train and test data even in scenes with high grid resolution. The proposed technique solves the following subproblems:

1) A space downscaling technique to maximize the efficiency of simulation space
2) A binarization technique for minimizing the numerical dissipation of density that occurs in progressive downscaling.
3) A technique to build a quadtree (octree in 3D) in a bottom-up style in a downscaled space and merge data
4) A method of collecting data necessary for training based on a quadtree built in a downscaled space.

Solving the first issue reduces the size of the space in which the quadtree needs to be computed (like a multigrid solver [23]), and progressive downscaling is possible as much as the user wants. If the second issue is solved, the small density is not lost in the process of reducing the space, and the original shape of the density distribution can be maintained even in a downscaled space. Solving the third problem allows us to calculate quadtree in a smaller space. By solving the last problem, quadtree can be upscaled to its original size to collect efficiently the data needed for training.

II. PRELIMINARY

A quadtree is a tree data structure with four children in each internal node and is an algorithm used to partition a two-dimensional space adaptively by dividing a 2D square space into four quadrants and subdividing it recursively according to a given criterion. The data associated with a leaf cell varies depending on the application, but a leaf cell generally has a ‘‘minimum unit of information of interest’’. The octree is an extended version of the quadtree applied to the 3D space, and the internal node is divided into eight octants and subdivides the cube-shaped space recursively. Our method uses a quadtree in 2D and an octree in 3D based on the data structure presented.

III. RELATED WORK

In this section, we review the latest studies on spatial partitioning and SR in smoke simulation.

A. HIGH-RESOLUTION SMOKE SIMULATION

Smoke simulation is an example of a typical grid-based simulation. Stam [40] solved the Navier–Stokes equation involved in smoke simulations using a semi-Lagrangian method. Some studies [9], [10] considered the turbulence effect to express flow details in LR smoke simulations. Bridson *et al.* [48] generated a turbulence velocity field by adding noise, and Zhang *et al.* [49] refined the details by restoring missing vorticity elements. In recent years, studies to perform HR smoke simulation using deep learning have appeared. Convolutional neural networks(CNNs) [1], [4] or generative adversarial networks(GANs) [3] can be used to achieve finely expressed simulations efficiently. Tompson *et al.* [2] and Xiao *et al.* [11] proposed using a CNN-based approach to solve the Navier–Stokes equation efficiently.

Some studies attempted to perform SR using previously acquired LR smoke simulation results rather than solving the simulation equation directly. The LR results received in these studies were upscaled to a higher resolution, and the coarse shape was refined in more detail.

CNNs and GANs are popular deep-learning methods for SR-related research. Dong *et al.* [30] proposed the first CNN-based solution for single-image SR. Ledig *et al.* [31] and Chu *et al.* [32] proposed GAN-based methods for image SR. Wang *et al.* [33] proposed an SR technique that magnifies an image eight times using a multiscale model based on CNNs and GANs. For smoke flow simulation, Liu *et al.* [45] employed a CNN for SR smoke image. Chu and Thuerey [12] used a CNN model to synthesize HR smoke flow. Xie *et al.* [28] and Werhahn *et al.* [47] employed a GAN for both 2D and 3D SR smoke flow. Bai *et al.* [46] used deep learning to perform SR that produced HR flow details with dynamic features.

Although these approaches can produce HR smoke flow results from input LR smoke flows, they are inefficient because they perform operations on the entire space, including spaces where valid data do not exist. To address this issue, we propose an accelerated SR method that partitions simulation data into regions with and without valid data based on a tree structure and performs the SR process only in regions with valid simulation data.

B. SPATIAL PARTITIONING

Quadtree [34] and octree [35] methods have been used widely for partitioning 2D and 3D spaces, respectively. Space partitioning methods are often used to increase efficiency by performing operations in only a few spaces with meaningful data in the entire target space. Whang *et al.* [44] employed an octree for optimizing the ray-tracing process. Shi *et al.* [43] used octrees to generate more detailed results than simulations conducted without octrees under the same conditions. Zhou *et al.* [36] proposed an algorithm for reconstructing a 3D surface model using a GPU-based octree. Recently, studies combining space partitioning and deep learning have emerged. Riegler *et al.* [37] and Wang *et al.* [38] proposed techniques for classifying, retrieving, and segmenting 3D models using octrees and CNNs. Tatarchenko *et al.* [39] addressed memory and speed issues by partitioning the space used for the 3D model generation. However, these studies have focused on modeling and rendering, rather than simulation.

IV. OVERVIEW

Our method consists of the following processes: In the preprocessing process, the smoke density data are divided into patches. Assuming that the divided patches are nodes of the smallest size, the quadtree is merged in a bottom-up style. In this process, downscaling and binarization are performed to prevent the dissipation of density data and reduce computational space. Then, SR is performed to convert the space into higher-resolution data. Our algorithm operates as follows (see Figure 3):

1) Data are preprocessed to construct a tree structure. The preprocessed data and the tree structure are used to partition the entire space adaptively and find areas requiring SR.
2) In the process of constructing the quadtree, downscaling and binarization are performed to reduce the space without numerical dissipation of the density.
3) After dividing the patches in the smallest downscaled space, we classify each cell into FD (Filled density) and ED (Empty density), and build a quadtree in a bottom-up style.

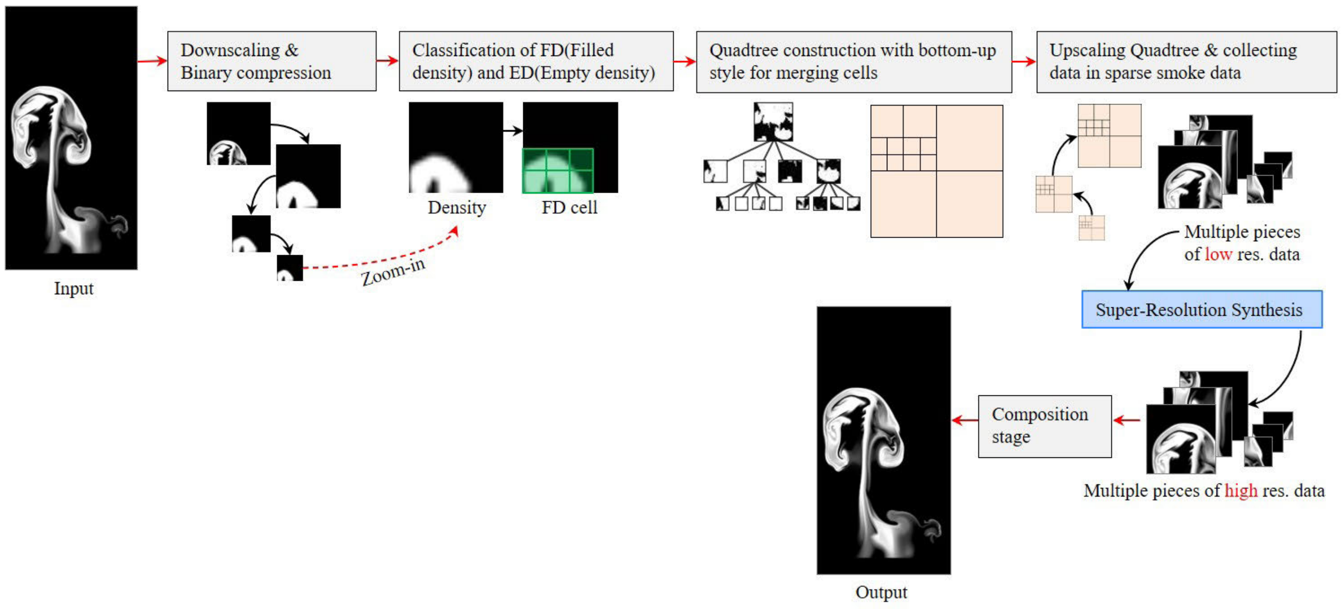

Input
Downscaling & Binary compression
Zoom-in
Classification of FD(Filled density) and ED(Empty density)
Density
FD cell
Quadtree construction with bottom-up style for merging cells
Upscaling Quadtree & collecting data in sparse smoke data
Multiple pieces of low res. data
Super-Resolution Synthesis
Multiple pieces of high res. data
Composition stage
Output

FIGURE 3. Algorithm overview.

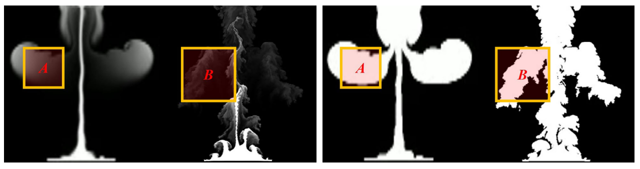
4) After the quadtree constructed from the downscaled space is upscaled to the original size, SR is performed using only the density data contained in the leaf nodes.
5) To convert the LR data to a higher resolution, the SR is performed using the training model using tempoGAN [28] and the partitioned data. Postprocessing is also required to merge the partitioned HR data.

In the next section, our method is described using quadtree and 2D simulation results.

V. INPUT DATA PARTITIONING BASED ON TREE STRUCTURE

Quadtree for 2D space and octree for 3D space have been widely used for spatial optimization. We also employ the quadtree (or octree) to partition the simulation space with the spatial information in the tree structure and the density data. In this process, if the smoke density is given as input data, it is compressed through binarization and downscaling. If the smoke density data are not dissipated after binarization, it is possible to optimize the performance of the operation that builds the quadtree. As shown in Figure 4b, even small density values can be captured without numerical loss because of the binarization process. Areas *A* and *B* of Figure 4a were clearly captured despite the high probability of losing their features because of their small density (see ***A***,***B*** in Figure 4b).

Then, the input data are divided according to the predetermined patch size. We construct a matrix of the maximum density values of each patch using the parallel computation of the GPU. Using the generated matrix, a state value that determines whether SR should be performed or not is assigned to each patch. Using the assigned state value and the information of the lowest node of each patch, we construct a tree structure in a bottom-up manner. The generated tree is used to find the area where SR should be performed. This process is shown in Figure 3, and its details are described in the following subsections. Unlike the classic quadtree, which divides patches from input simulation data, our method goes through downscaling and binarization, so we divide the patches in the smallest downscaled space. This process is also described in more detail later.



(a)Original density data (left : 128×128, right : 512×512) (b)Binarized density data (left : 128×128, right : 512×512)

FIGURE 4. Binarization of density data.

A. INPUT DATA PREPROCESSING

The input data are composed of the density value of each grid point. First, the entire simulation space is divided into smaller sizes in the preprocessing step. As a tree structure to store divided spaces, we use the quadtree (for 2D space) or the octree (for 3D space).

Through several experiments, we confirmed that it was almost invisible when the density value decreased to less than 0.01. Based on this result, an experiment was conducted by setting threshold sets (0.015, 0.01, 0.005) with values near 0.01. When the threshold was set to 0.015, the FD path was incorrectly judged as ED in some scenes, and when it was 0.005, the results were similar to those of 0.01, but the ED area that does not require computation was judged as FD, and

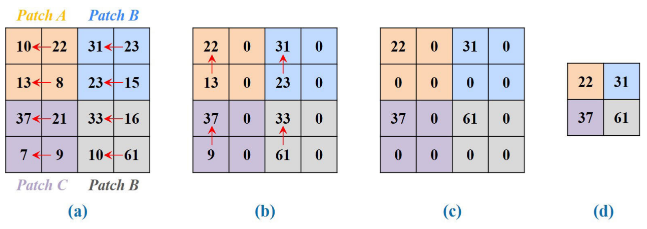


FIGURE 5. Calculation of the maximum density value matrix for patches.

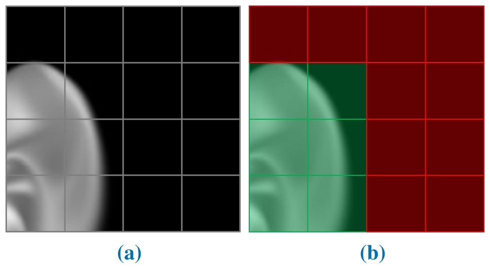


FIGURE 6. State values for partitioned patches of input data. The green patches were set to "FD," and the red ones were set to "ED".

the performance slowed because of over-computing. Therefore, in this paper, 0.01 was set for all results, and in this case, the most stable result between performance and quality was obtained.

B. STATE VALUES FOR INPUT DATA PATCHES

Based on the state value of the input data, we can find the area that needs the SR. State values are specified as follows.

First, the divided input data are partitioned into patches of a predetermined size, and the maximum density value of each patch is calculated and stored in a matrix via GPU-based parallel operations.

Next, the state value of each patch is determined by comparing the maximum value of the matrix with the predefined threshold. If the maximum density value of each patch exceeds the threshold, its state is set to the "FD(Filled Density)" requiring SR processing. In the opposite case, the patch is set to "ED(Empty Density)" where SR is not performed (see Figure 6).

If the threshold is too high, many details will disappear, and if it is too low, SR will be performed in dim areas that are not recognized by the eye, resulting in lower overall performance. In this paper, we set the threshold value of 0.01 through several preliminary experiments. Section VII presents the results of the performance improvement from the GPU-based method we used.

C. BINARIZATION AND DOWNSCALING OF SIMULATION SPACE

When we compress and downscale the data to reduce computation time, we reduce the size to $\frac{1}{4}$ times ($\frac{1}{8}$ times in 3D) compared with the previous size. When downscaling, the data were compressed by calculating the average density of four adjacent nodes. In this paper, downscaling was performed three times for each experiment (see Figure 7).

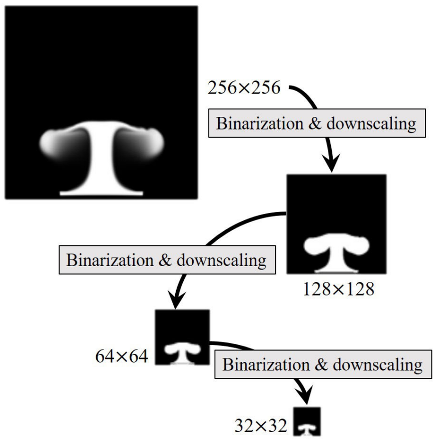

256×256
Binarization & downscaling
128×128
Binarization & downscaling
64×64
Binarization & downscaling
32×32

FIGURE 7. An example of density compression by binarization and downscaling.

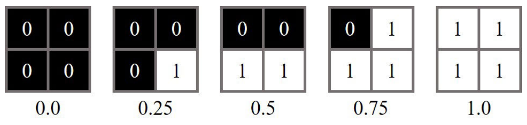


FIGURE 8. Binarized density patterns in 2D.

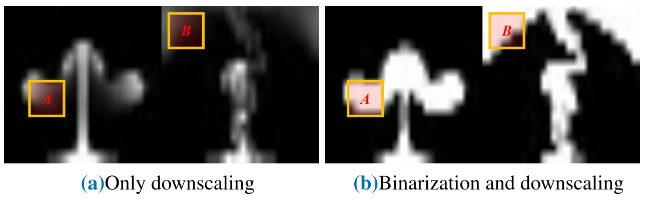


FIGURE 9. Change of density after combining progressive binarization with downscaling (grid res.: 32 × 32).

To determine the threshold to be used when performing downscaling for the first time on the original data, we performed binarization using values such as 0, 0.01, 0.1, and 0.05. As a result of repeated experiments in various environments, the quality of the results was the best when the threshold α was set to 0.05. Therefore, in the experimental results of this paper, the downscaling was performed by setting 0.05 as the α: $density > \alpha$? 1: 0. When performing the second and third downscaling, the target data consist of only 0s and 1s because binarization has already been performed once. In this case, the average density of four adjacent nodes is one of 0, 0.25, 0.5, 0.75, and 1, and the threshold for binarization at this stage is set to 0.5 (see Figure 8). By minimizing data loss through this binarization process and reducing the size of data through downscaling, the amount of computation required for quadtree operation has been greatly reduced.

Figure 9 shows the change in density after combining binarization with downscaling. In the case of only downscaling, as the space decreases, the smaller density value vanishes and

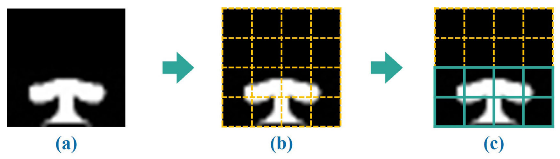


FIGURE 10. Classification of FD and ED according to the presence or absence of density: (a) binarized and downscaled data, (b) patch split, (c) classification of FD and ED (cyan: FD node).

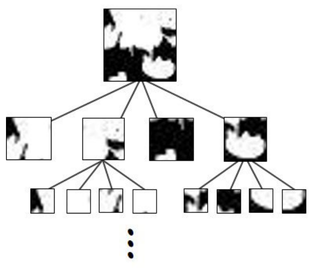
gradually converges to 0 (see ***A*,*B*** in Figure 9a). In contrast, when binarization is combined with downscaling, the results show that the detailed characteristics of the area in which the density exists are well maintained (see ***A*,*B*** in Figure 9b).

Unlike the classic quadtree, which splits the patches at the original resolution, when applying a quadtree including downscaling and binarization, they are split in the smallest downscaled space. In this paper, the size of the patch was set to 16×16, and nodes were created based on data compressed three times. To merge the quadtree in a bottom-up style using the generated nodes, we first determine whether density exists in each node, compare it with the threshold, and classify it into FD and ED (see Figure 10).

D. CREATING THE FINAL TREE STRUCTURE

As described earlier, we first create the lowest nodes and merge them in a bottom-up manner to build a quadtree. The lowest nodes have specified patch-state values and the patch-state value of the parent node is determined according to the patch-state values of the child nodes, as shown in Figure 12a.

Each node of the tree has *data*, *key*, and *state* values (see Figure 11). *data* indicates the density value of the node. *key* has the x-coordinate and y-coordinate (including the z-coordinate in the case of octree) indicating the location of the node, and the tree depth used when constructing the tree (see Figure 12c). The depth and position are used when merging the result after the SR process is completed. In the case of *state*, it can be FD, ED, and MIX values.



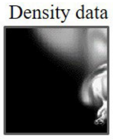
Each node information	
Data	Density data 
Key	Depth Position : x, y, z
State	FD or ED or MIX

FIGURE 11. Information at each node of the quadtree.

In this paper, we assign the density of each patch as *data* of the lowest node (see Figure 12a). Among the *key* values of the lowest node, the depth is calculated using Equation 1.

$$d = log_2(D_{width}/N_{width}), \tag{1}$$

where d is the depth of the current node, D_{width} is the width of the entire input data, and N_{width} is the width of the current node. Then, we create a parent node in bottom-up style with four nodes (eight in the case of octree) (see Figure 12b). When creating a parent node, *data* becomes the sum of the *data* of child nodes, the depth of the *key* value is reduced by 1, and the position value is obtained by merging all child nodes. Finally, the state values are determined by checking the state values of child nodes as follows: If the state values of all child nodes are the same, that value is assigned. If FD and ED are mixed in the state values of all child nodes, MIX is assigned.

If all child nodes have the same *state* value, they are deleted. In the case of ED, SR is not required, and in the case of FD, it is faster to perform SR once for the parent node than for each child node. If this operation is repeated until reaching the root node, a tree with state values assigned to all nodes is finally created (see Figure 12c). After the tree is created, the *state* value is checked from the root node to collect *data* and *key* values from all FD nodes (see Figure 13). The ED node is removed because it does not require SR. After traversing all leaf nodes, the collected data are transferred as input data of SR.

Figure 14 shows examples of quadtrees constructed in downscaled space using binarized density. By extracting the area with density through quadtree and using the multiscale technique, the amount of calculation was reduced, and density dissipation was prevented through binarization. As shown in the orange boxes of Figure 14, although the density is relatively small, it is clearly expressed by binarization, so that the node is divided.

Figure 15 shows an overview of the entire process explained earlier. First, leaf nodes divided into patches are compared with the threshold and classified into FD and ED (see Figure 15a). The quadtree calculated in downscaled space is upscaled (see Figure 15b) and postprocessing is performed according to the size of the original data. In this process, only the original density data corresponding to the leaf node of the upscaled quadtree are used in the neural network. SR is performed only in the area where the density exists, and the area without the density is filled with black and composited (see Figure 15c).

Although the quadtree is calculated in a downscaled space, it is stably upscaled to the size of the original data, and the amount of input data is reduced using only the data in the leaf nodes to the neural network (see Figure 16).

VI. SUPER-RESOLUTION(SR) PROCESSING

A. SR MODEL

The deep-learning model we used to perform the SR operation is tempoGAN, the state-of-the-art model proposed by Xie *et al.* [28] suitable for performing smoke SR. We select nodes with FD state values in the quadtree and provide them as input to tempoGAN. The tempoGAN model we used consists of one generator and two discriminators. The generator consists of two nearest neighbor layers and four residual blocks. The two discriminators are the space discriminator

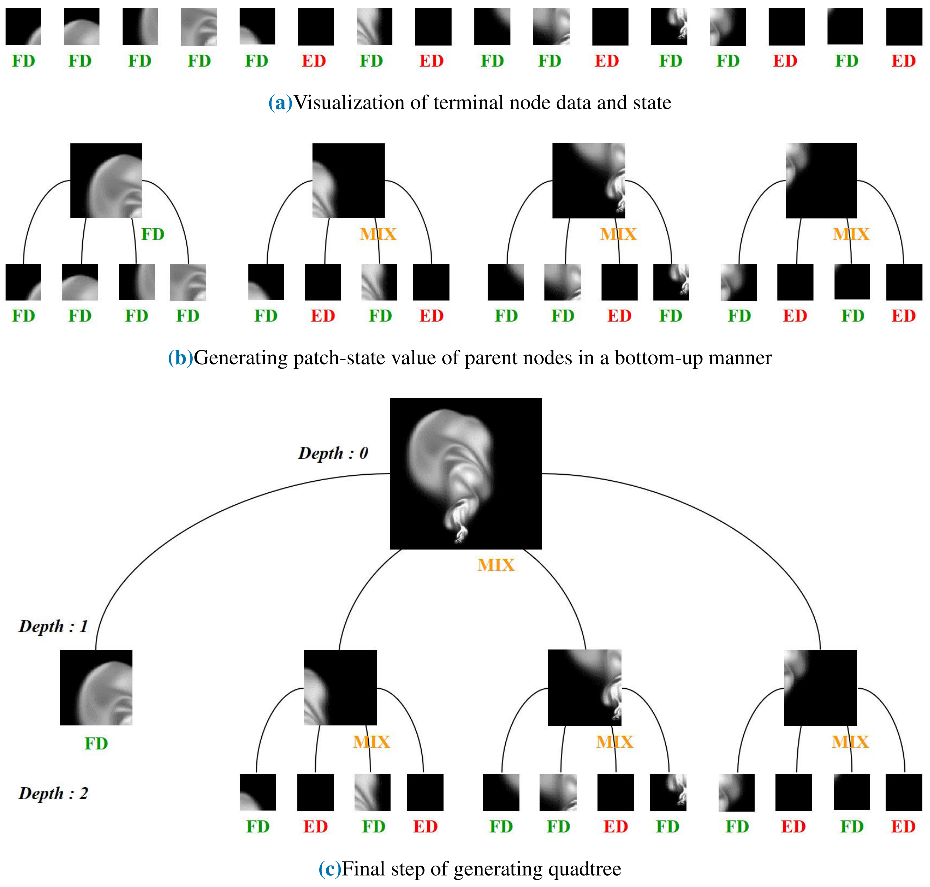

FD
ED
MIX
(a)Visualization of terminal node data and state
(b)Generating patch-state value of parent nodes in a bottom-up manner
Depth : 0
Depth : 1
Depth : 2
(c)Final step of generating quadtree

FIGURE 12. Overview of generating the quadtree containing patch-state values.

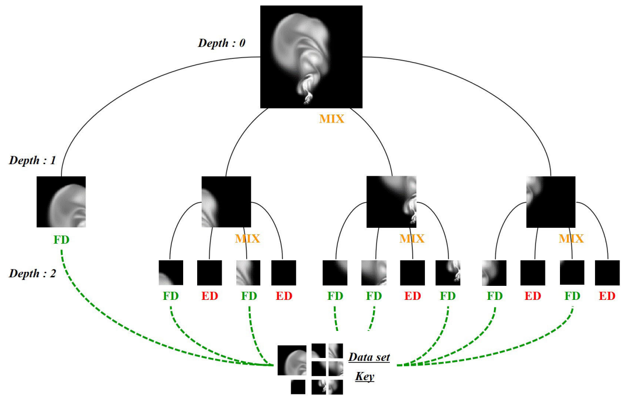


FIGURE 13. An example of the quadtree nodes for SR.

and the temporal discriminator, both of which are composed of four convolution layers. The space discriminator receives the fake data created by the generator and the real data obtained through the simulation at the frame t and determines whether they are real or fake (see Space Discriminator in Figure 17). The temporal discriminator receives the data created by the generator and the real data obtained by simulation at frames $t-1$, t, and $t+1$ as inputs and determines whether they are fake or real (see Temporal Discriminator in Figure 17). In this paper, after learning of the model is

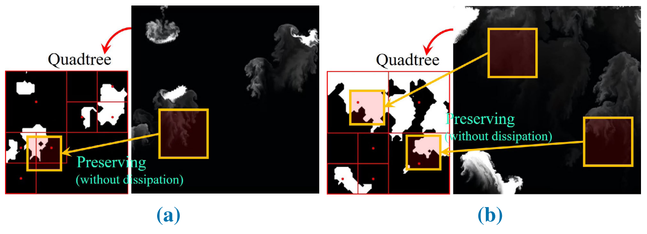


FIGURE 14. An example of downscaled quadtree using binarized density.

completed in this way, SR was performed using the generator of this model.

The tempoGAN data for SR where used by downloading the trained model from the tempoGAN author's GitHub. Because the tempoGAN algorithm divides large-sized data into multiple patches of the same size and performs SR, the generator model can be applied to a single-size patch. However, because this paper uses data sets of various sizes generated through trees, the generator model of tempoGAN has been modified so that it can receive inputs of various sizes. So, pieces of various sizes are input to the generator model of tempoGAN, and as a result, a data set of multiple pieces of HR is obtained.

B. POSTPROCESSING

After the HR data are produced by the SR operation, the partitioned data need to be merged to recreate the entire scene.

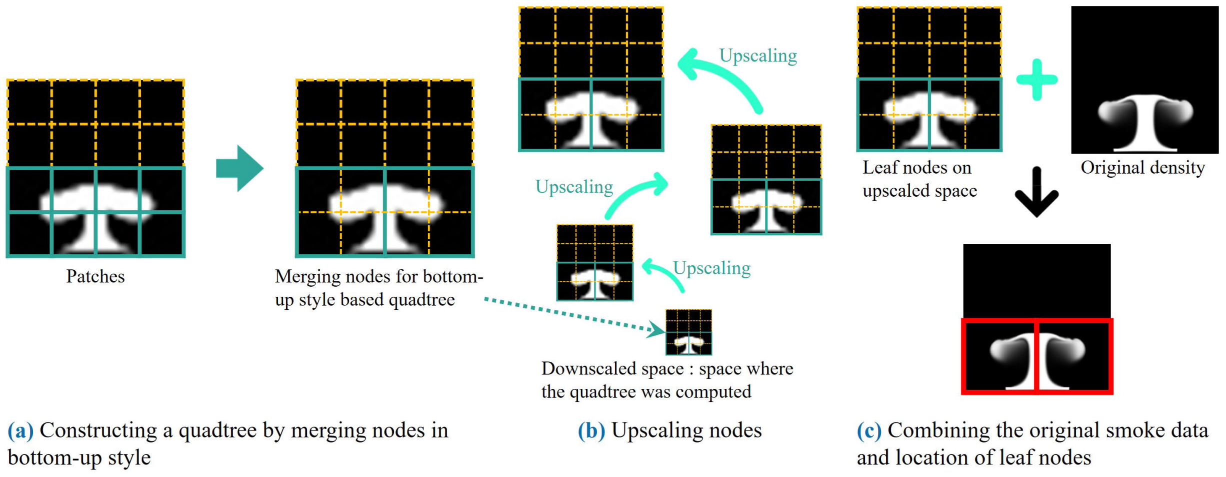

Patches
Merging nodes for bottom-up style based quadtree
Upscaling
Upscaling
Upscaling
Downscaled space : space where the quadtree was computed
Leaf nodes on upscaled space
Original density
(a) Constructing a quadtree by merging nodes in bottom-up style
(b) Upscaling nodes
(c) Combining the original smoke data and location of leaf nodes

FIGURE 15. An overview of quadtree construction and upscaling.

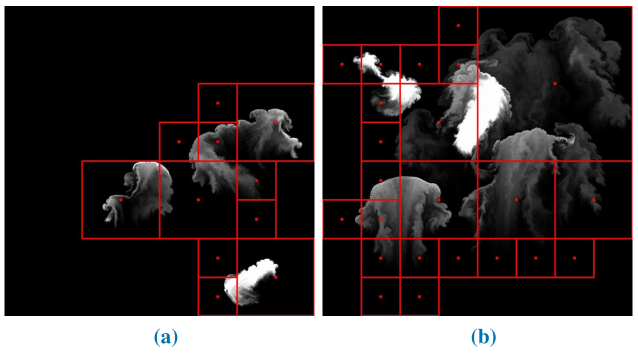


FIGURE 16. Examples of quadtrees calculated in downscaled space upscaled to its original size (original grid res.: 512 × 512, red box: leaf node).

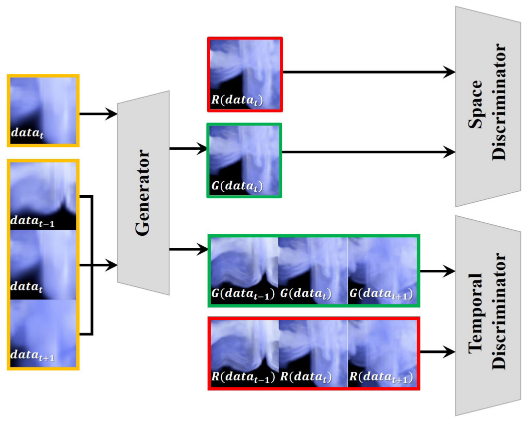


FIGURE 17. Network architecture of tempoGAN [28] (*R*: real data, *G*: generated data by the generator model).

As described in Section V-B, each node stores *data* and *key* information. Because the location of the node and the size of the patch are fully specified from the *key* with the depth and location value, data of patches with different sizes can be merged.

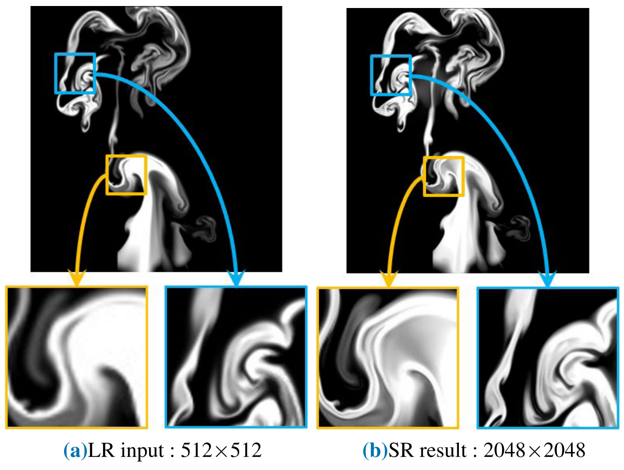


FIGURE 18. 2D SR processing.

Before merging the patches we first create an empty space with the size of the target scene. Then, the final result is completed by synthesizing the data at the corresponding patch location according to the *key* information of the post-SR data.

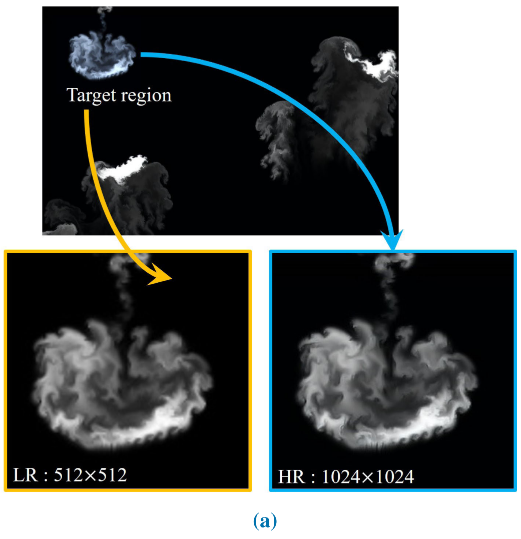
VII. EXPERIMENTS AND ANALYSIS

In this paper, all experiments were performed using a standard PC with an Intel Core i9 X-series CPU and a GTX Titan RTX 24GB graphics card. The smoke simulation was implemented using the C++ language. We used the deep-learning model proposed by tempoGAN [28] for the SR process.

A. RESULTS OF SR PROCESSING

1) 2D SCENES

The 2D SR results are shown in Figure 18. From the input data of 512 × 512 resolution, we obtained the result of 2048 × 2048 resolution through SR processing. The HR details are clearly visible in certain areas (blue and yellow boxes in Figure 18). When this scene was performed using the original tempoGAN without quadtree, it took an average



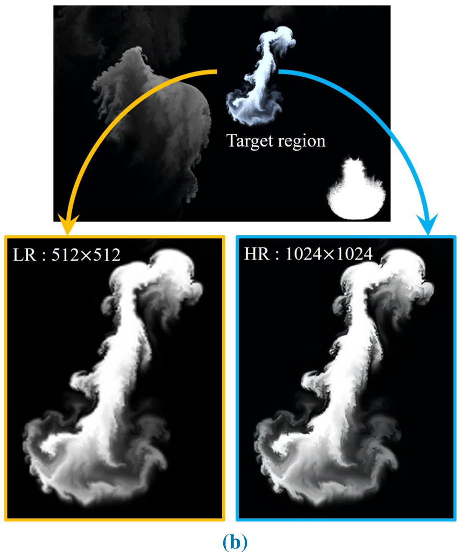
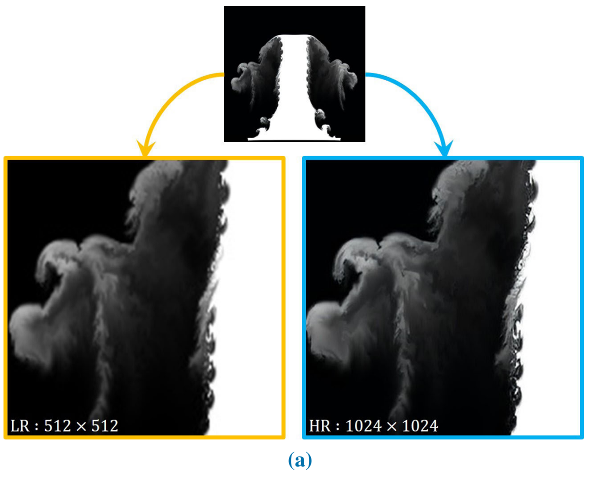


FIGURE 19. Smoke-injecting simulation.



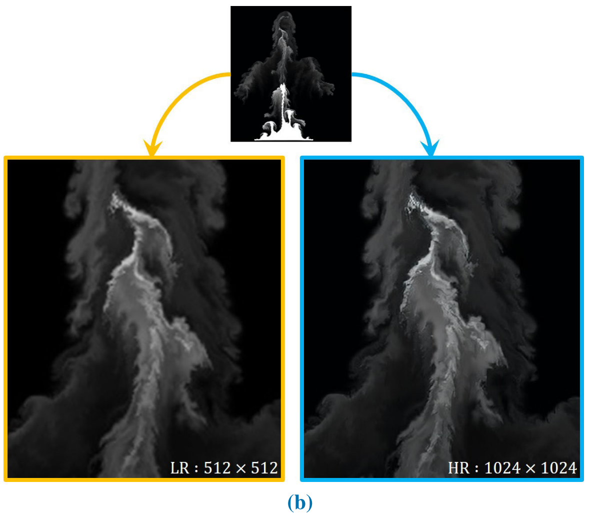


FIGURE 20. Rising smoke simulation.

of 1.3 seconds, but when the proposed method with quadtree was used, it took 0.025 seconds. Our method significantly improved the detail and clarity of the features when compared visually with LR smoke (see Figure 18b).

We also experimented in an environment where smoke is randomly injected (see Figure 19). The input smoke data have a resolution of 512×512, and are upscaled by two times. When using the original tempoGAN technique, it took an average of 1.2 seconds, but when using our technique, the performance was approximately double. Because quadtree has proven its efficiency in terms of speed and memory because of its data structure nature, we conducted additional experiments. When using only the classic quadtree without downscaling and binarization, it took about 0.6 seconds, but our method, including downscaling and binarization, took 0.04 seconds. In the scene in Figure 19, the performance is improved by about 15 times compared with the classic quadtree. As in the previous experiments, we were able to obtain clear results in terms of quality.

We also conducted experiments in a rising smoke environment. In our results shown in Figure 20a, not only the density was clearly expressed, but the turbulent pattern was clearly well expressed in the vicinity of the free surface in contact with air. The results show that similar results are well expressed in Figure 20b.

2) 3D SCENES

Figure 21 shows the three training scenes used to perform the 3D simulation. Figure 22 shows the result of $512 \times 256 \times 256$ resolution obtained from the input data of $128 \times 64 \times 64$ resolution. As shown in the figure, the blurred areas are expressed in great detail after the SR operation.

One of our goals in this study is to improve the speed of SR calculation while maintaining the visual quality when using the original tempoGAN technique. We devised an equation representing the quality difference between the results using the original tempoGAN and our results as follows:

$$E = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|, \tag{2}$$

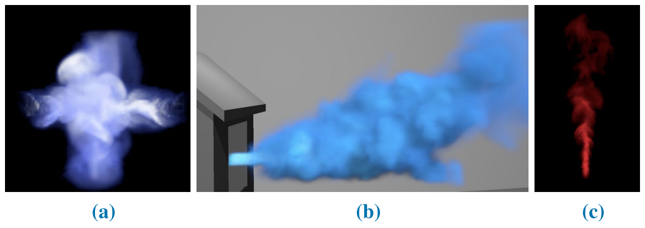


FIGURE 21. The three training scenes used for 3D smoke simulation: (a) collision(128 × 64 × 64), (b) boost(128 × 64 × 64), (c) basic(64 × 128 × 64).

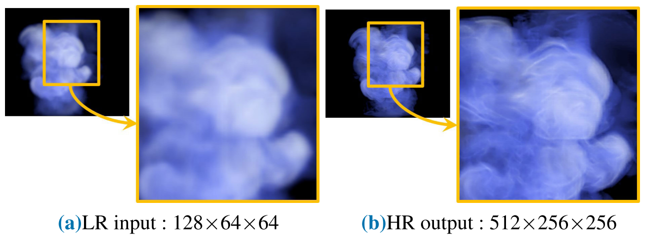


FIGURE 22. Comparison of before (a) and after (b) 3D SR processing.

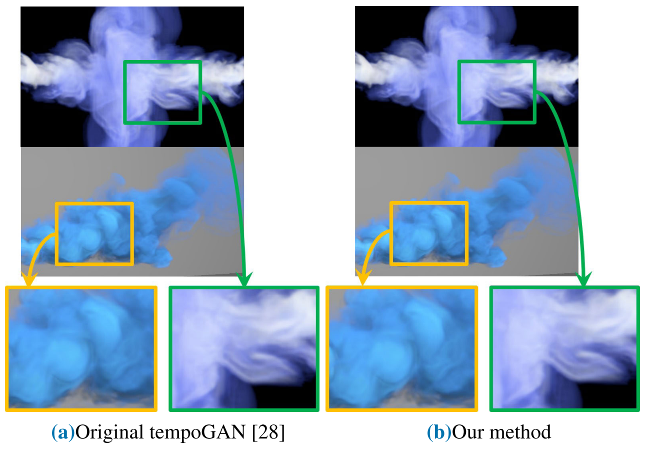


FIGURE 23. Comparison of results between original tempoGAN and our method. The average error values for Figure 21a(collision) and Figure 21b(boost) were 0.000058 and 0.000065, respectively.

where E is the error value, N is the total number of frames, y_i is the HR data generated by tempoGAN for frame i, and $\hat{y}_i$ is the HR data generated by our method. The overall error is computed by averaging the sum of the individual errors over all frames. The error value for two different scenes is shown in Figure 23, showing that the difference between the HR results from our method and the tempoGAN method is almost zero, with the close-up details (green and yellow boxes) confirming the similarity. We compared y_{tempo} and y_{ours}, which are results from the same input data, to determine how different our method is from the original tempoGAN. The mean error was calculated using the absolute value of the difference between y_{tempo} and y_{ours} for all grids.

As shown in Figure 24, despite using the adaptive octree, the density of SR smoke looks lossless and the pattern is clearly expressed.

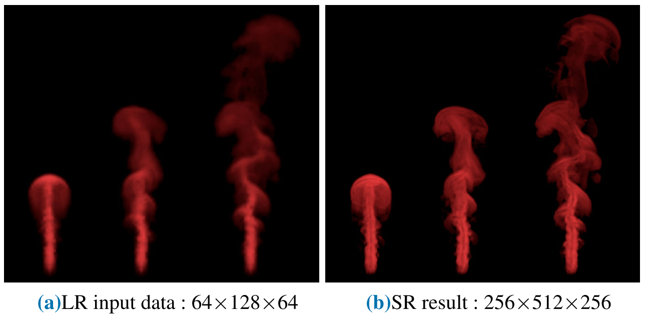


FIGURE 24. Results of 3D SR processing (Figure 21c).

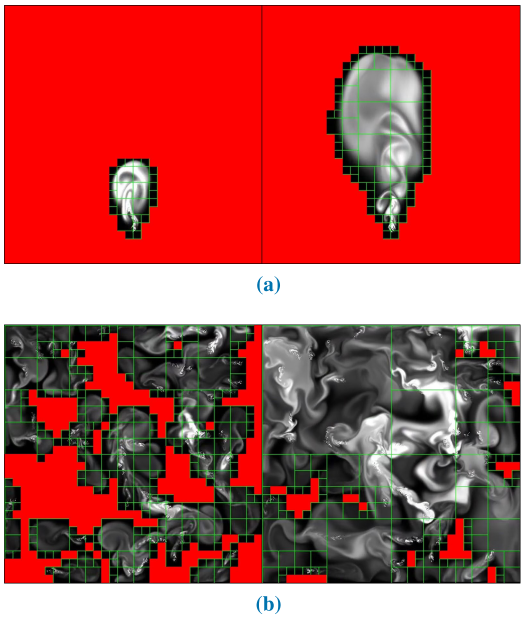


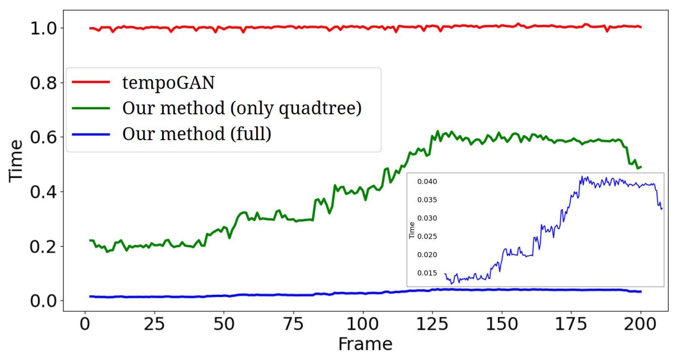
FIGURE 25. Adaptivity test for rising smoke (red cell: empty region).

Our method improves the efficiency in terms of speed and memory by applying the SR operation to the smoke simulation based on the adaptive grid (e.g., octree). Although our method does not represent realistic elements of the smoke movement, integrating our method into existing detail-enhancement studies can improve details and efficiency [5], [6], [50]. Some studies improve detail using deep learning for fluid simulation. They proposed a style-transfer network that can control the detailed motion of smoke while using a grid-based structure. Although this technique is not an SR technique, it is the first study to apply image-style-transfer research, which has been studied a large amount in computer vision, to a physics-based simulation. Because this study also uses grids and densities like ours, it can be easily integrated based on octrees.

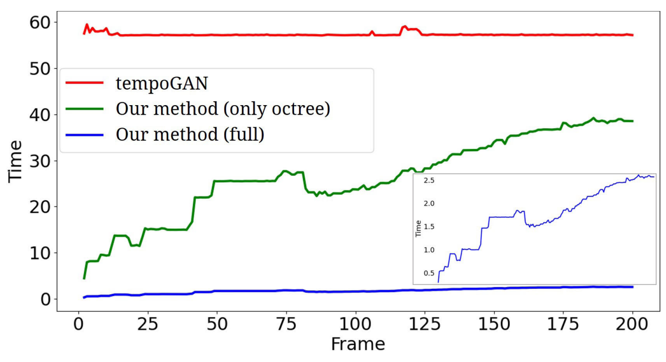
B. DISCUSSION

1) SPATIAL ADAPTIVITY

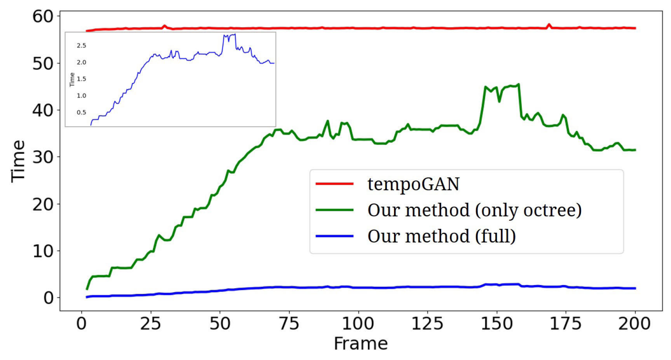
Two scenes were created to determine the efficient of the proposed method in terms of space. We experimented by upscaling the quadtree calculated in the downscaled space to its original size (see Figure 25).



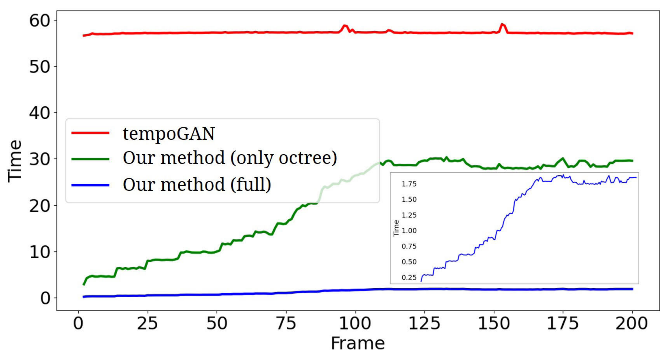
(a)2D scene (Figure 18)



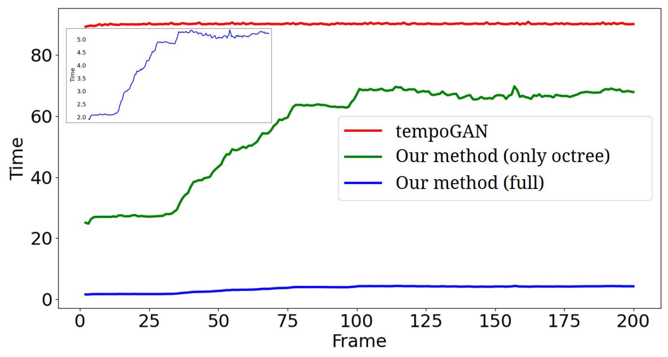
(b)3D collision scene (Figure 21a)



(c)3D boost scene (Figure 21b)

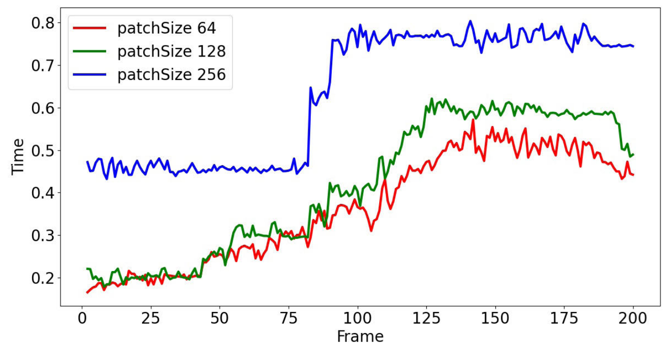


(d)3D basic scene (Figure 21c)

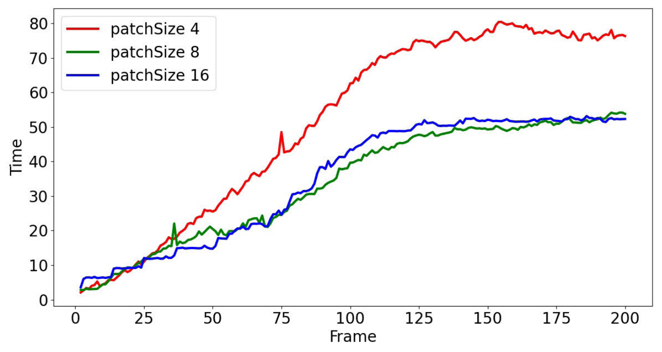


(e)Smoke text scene (Figure 28)

FIGURE 26. Comparison of execution time between the original tempoGAN and our method (inset image: our method(full)).



(a)2D scene (512×512)



(b)3D basic scene (Figure 21c, 64×128×64)

FIGURE 27. Comparison of execution times for different patch sizes.

Figure 25a shows the adaptivity of our technique through an example of rising smoke. The quadtree is partitioned only in the area where the density exists, and the SR operation is not performed on the empty area without the density.

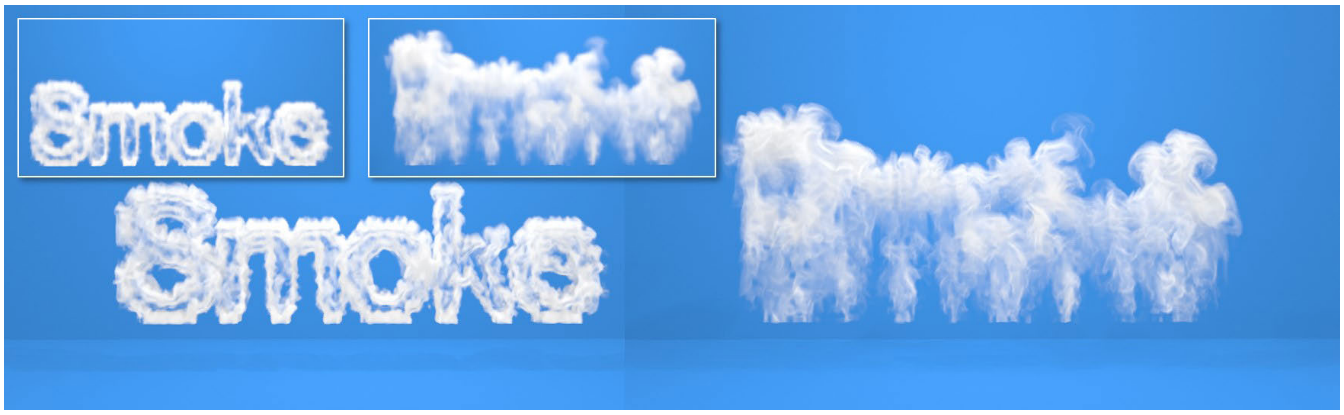


FIGURE 28. SR results for the smoke text scene generated by our method (inset image: LR results).

We randomly distributed the location where the density is sourced, and then performed a smoke injection simulation. At the beginning of the simulation, the quadtree was divided only near the sourcing position, but as the density spreads, the area in which the quadtree is calculated also expands (see Figure 25b). As the computational space increases, the number of nodes merged increases, so the total number of nodes does not continue to increase.

2) COMPARISON WITH THE PREVIOUS METHOD

First, we compared the execution time between our method and the original tempoGAN. Because the original tempoGAN technique applies the SR operation to the entire simulation space, the execution time for each frame does not change significantly. In contrast, our method has a large variation in execution time for each frame because there are many changes in the quadtree nodes for each frame.

Figure 26a shows the results of performing the SR operation on 2D input data with a patch size of 128 and a resolution of 512 $\times$ 512. When using the original tempoGAN, it took about 1.2 seconds, and when using only quadtree among our methods(our method(only quadtree) in Figure 26a), it took about 0.5 seconds on average. When we used our method with only quadtree, the execution time was 5 times faster for simple scene frames, and about 1.6 times faster for dynamic scene frames. On average, the frames are processed more than 3 times faster. As described, when the quadtree was combined with downscaling and binarization (our method(full) in Figure 26a), the performance was improved by about 16 times compared with the classic quadtree alone.

Similar patterns appear in other scenes in Figure 26. If only quadtree is used, the performance improvement decreases toward the latter half of the simulation, where the scene becomes more complex and dynamic, but when downscaling and binarization are combined, the performance improves dramatically in every frame. Figure 26c and 26d, showing the boost and basic scenes, have the same maximum acceleration factor at the beginning.

3) DIFFERENT PATCH SIZES

Because the simulation space is divided into patches, the overall simulation performance is inevitably affected by the size of the patch. Because the time for SR processing in our proposed method depends on the complexity of the scene and the patch size, it is necessary to specify a suitable patch size. Figure 27 shows the SR execution times when using three different patch sizes for the 2D and 3D basic scenes.

For a 2D basic scene with input data resolution of 512 $\times$ 512, we tested with patch sizes set to $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{8}$ of the input data resolution. For a 3D basic scene with input data resolution of 64 $\times$ 128 $\times$ 64, we tested with patch sizes set to $\frac{1}{4}$, $\frac{1}{6}$, and $\frac{1}{8}$ of the input data resolution. According to our experiments, the optimal patch size is $\frac{1}{4}$ (for 2D) or $\frac{1}{8}$ (for 3D) of the input data size. In this paper, through various experiments, the patch size for the 3D scene was set to 16 and a good result was produced.

Finally, we compared the execution time for each of the 3D scenes using a patch size of 16. Figure 26 demonstrates that our method performs SR faster than tempoGAN in each frame, and the error value of 0.000097 is insignificant.

4) DIFFERENT SCENES

As mentioned earlier, the more complex and dynamic the scene, the longer the execution time will take. In this paper, experiments were conducted on various scenes according to various situations and settings, and the configuration and performance of each scene are summarized in Table 1. The parameters in this paper are summarized in Table 2.

Because our method applies the SR operation only to patches with a state value of FD or MIX, the more complex scene in the second half of the simulation (with more variable-density data) requires more processing. By contrast, the scene at the start of the simulation required little processing, because there were almost no variable-density data. Figure 28 shows the SR results generated by our method of the target-driven smoke scenes.

TABLE 1. Simulation configuration and performance.

Figure	Input size	Output size	Total frame	Performance Improvement (avg.)
18	512×512	2048×2048	200	Ours(only quadtree) / tempoGAN: ×3 Ours(full) / Ours(only quadtree): ×16
21a	128×64×64	512×256×256	200	Ours(only octree) / tempoGAN: ×2.5 Ours(full) / Ours(only octree): ×13
21b	128×64×64	512×256×256	200	Ours(only octree) / tempoGAN: ×2.5 Ours(full) / Ours(only octree): ×16
21c	64×128×64	256×512×256	200	Ours(only octree) / tempoGAN: ×2.5 Ours(full) / Ours(only octree): ×16
28	128×64×64	512×256×256	200	Ours(only octree) / tempoGAN: ×1.8 Ours(full) / Ours(only octree): ×13

TABLE 2. Simulation parameters. We used these specific parameters to generate the example animations shown in this paper.

Name	Description	Value
α	Binarization threshold	0.05
Data	Density data	–
Key	Depth, position(x, y, z)	–
State	FD ED or MIX	–
Patch size	2D input data	$\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{8}$ of the input data res.
Patch size	3D input data	$\frac{1}{4}$, $\frac{1}{6}$, and $\frac{1}{8}$ of the input data res.

5) MEASURING EXECUTION TIME

The execution times of the original tempoGAN [28] and our method were measured and compared. Because tempoGAN uses data simulated in the entire space without a quadtree, we measured the average SR time spent for training and testing. When measuring the execution time of our method using only the quadtree, the time taken to build the quadtree and the average SR time taken for training were summed. When measuring the execution time of our method in which all operations were applied, the time required for binarization and downscaling was added.

VIII. CONCLUSION AND FUTURE WORK

We propose an accelerated SR method for a 3D smoke scene based on an octree technique. While most of the existing SR methods perform operations for the entire simulation space, our method can reduce the execution time because it searches for a region to which the SR is to be significantly applied. We first divide the space based on the presence of density data, and this information is stored in a tree structure such as a quadtree or octree. Because of the elimination of unnecessary SR operations, the performance of our method is significantly improved compared with using the original tempoGAN [28]. Because the space partitioning technique in this study is widely used, it can be easily applied when performing SR operations for other types of fluid simulation techniques (e.g., Sato *et al.* [42] and Kim *et al.* [5], [6]).

The limitation of our study is that the optimal patch size can only be obtained empirically. In addition, the data optimization method proposed in this paper is a tree-based method that depends on the density of smoke. In other words, there is a limit to maximizing efficiency because it depends on the presence or absence of density, not adaptivity according to the level of detail of density. In addition, the adaptivity of data was not considered in the network configuration stage.

In the future, we will study how to determine the optimal patch size based on the given data. In addition, we will newly define the standard for adaptivity according to the level of detail, and study how to improve the performance in both data and network configuration processes.

In future work, we aim to investigate the automatic determination of suitable patch sizes, which would improve the efficiency of our method. Because our method of partitioning the simulation space to be operated on is a generalizable process, our approach should be applicable to the acceleration of other fluid-simulation-based SR models or style-transfer models (e.g., Sato *et al.* [42] and Kim *et al.* [5], [6]).

REFERENCES

[1] Y. LeCun, E. B. Boser, S. J. Denker, D. Henderson, E. R. Howard, E. W. Hubbard, and D. L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[2] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating Eulerian fluid simulation with convolutional networks," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 3424–3433.
[3] J. Ian Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, C. A. Courville, and Y. Bengio, "Generative adversarial networks," *ArXiv*, vol. abs/1406.2661, 2014.
[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[5] B. Kim, V. C. Azevedo, M. Gross, and B. Solenthaler, "Lagrangian neural style transfer for fluids," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 1–52, 2020.

[6] 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, p. 188, 2019.

[7] R. Narain, J. Sewall, M. Carlson, and M. C. Lin, "Fast animation of turbulence using energy transport and procedural synthesis," *ACM Trans. Graph.*, vol. 27, no. 5, p. 166, 2008.

[8] T. Kim, N. Thürey, L. D. James, and H. M. Gross, "Wavelet turbulence for fluid simulation," *ACM Trans. Graph.*, vol. 57, p. 50, 2008.

[9] 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.

[10] B. Kim, Y. Liu, I. Llamas, and J. Rossignac, "FlowFixer: Using BFECC for fluid simulation," in *Proc. Eurograph. Workshop Natural Phenomena*, 2005.

[11] X. Xiao, Y. Zhou, H. Wang, and X. Yang, "A novel CNN-based Poisson solver for fluid simulation," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 3, pp. 1454–1465, Mar. 2020.

[12] M. Chu and N. Thürey, "Data-driven synthesis of smoke flows with CNN-based feature descriptors," 2017, *arXiv:1705.01425*. [Online]. Available: http://arxiv.org/abs/1705.01425

[13] H. Park, R. Yu, Y. Lee, K. Lee, and J. Lee, "Understanding the stability of deep control policies for biped locomotion," 2020, *arXiv:2007.15242*. [Online]. Available: http://arxiv.org/abs/2007.15242

[14] J. Won, J. Park, and J. Lee, "Aerobatics control of flying creatures via self-regulated learning," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–10, Jan. 2019.

[15] 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.

[16] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE Trans. Vis. Comput. Graphics*, vol. 26, no. 11, pp. 3365–3385, Nov. 2020.

[17] Y.-S. Sun, L. Wan, Y. Gan, J.-G. Wang, and C.-M. Jiang, "Design of motion control of dam safety inspection underwater vehicle," *J. Central South Univ.*, vol. 19, no. 6, pp. 1522–1529, Jun. 2012.

[18] X. Liu, S. Zhou, Y. Gao, H. Hu, Y. Liu, C. Gui, and S. Liu, "Numerical simulation and experimental investigation of GaN-based flip-chip LEDs and top-emitting LEDs," *Appl. Opt.*, vol. 56, pp. 9502–9509, 2017.

[19] A. W. Bargteil, T. G. Goktekin, J. F. O'Brien, and J. A. Strain, "A semi-lagrangian contouring method for fluid simulation," *ACM Trans. Graph.*, vol. 25, no. 1, pp. 19–38, Jan. 2006.

[20] T. Kim, N. Thürey, D. James, and M. Gross, "Wavelet turbulence for fluid simulation," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–6, Aug. 2008.

[21] 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.

[22] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating water and smoke with an octree data structure," in *Proc. ACM SIGGRAPH Papers*, 2004, pp. 457–462.

[23] A. McAdams, E. Sifakis, and J. Teran, "A parallel multigrid Poisson solver for fluids simulation on large grids," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation*, 2010, pp. 65–73.

[24] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," *Comput. Graph. Forum*, vol. 38, no. 2, pp. 59–70, May 2019.

[25] M. Shah, J. M. Cohen, S. Patel, P. Lee, and F. Pighin, "Extended galilean invariance for adaptive fluid simulation," in *Proc. ACM SIGGRAPH/Eurograph. Symp. Comput. Animation (SCA)*, 2004, pp. 213–221.

[26] M. Oshita and A. Makinouchi, "A dynamic motion control technique for human-like articulated figures," *Comput. Graph. Forum*, vol. 20, no. 3, pp. 192–203, Sep. 2001.

[27] S. Azadi, M. Fisher, V. Kim, Z. Wang, E. Shechtman, and T. Darrell, "Multi-content GAN for few-shot font style transfer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7564–7573.

[28] Y. Xie, E. Franz, M. Chu, and N. Thuerey, "tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow," in *Proc. ACM SIGGRAPH*, 2018.

[29] J. Lee, J. Won, and J. Lee, "Crowd simulation by deep reinforcement learning," in *Proc. 11th Annu. Int. Conf. Motion, Interact., Games*, Nov. 2018, pp. 1–7.

[30] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2016.

[31] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 105–114.

[32] M. Chu, Y. Xie, L. Leal-Taixé, and N. Thürey, "Temporally coherent GANs for video super-resolution," *ArXiv*, vol. abs/1811.09393, 2018.

[33] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers, "A fully progressive approach to single-image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 977–97709.

[34] G. M. Hunter and K. Steiglitz, "Operations on images using quad trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 2, pp. 145–153, Apr. 1979.

[35] D. Meagher, "Octree encoding: A new technique for the representation," Tech. Rep., 1980.

[36] K. Zhou, M. Gong, X. Huang, and B. Guo, "Data-parallel octrees for surface reconstruction," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 5, pp. 669–681, May 2011.

[37] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6620–6629.

[38] P. S. Wang, Y. Liu, Y. X. Guo, C. Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, p. 72, Jul. 2017.

[39] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2107–2115.

[40] J. Stam, "Stable fluids," in *Proc. ACM SIGGRAPH*, 1999.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[42] S. Sato, Y. Dobashi, T. Kim, and T. Nishita, "Example-based turbulence style transfer," *ACM Trans. Graph.*, vol. 37, no. 4, p. 84, 2018.

[43] L. Shi and Y. Yu, "Visual smoke simulation with adaptive octree refinement," *Comput. Graph. Imag.*, 2004, pp. 13–19.

[44] K.-Y. Whang, J.-W. Song, J.-W. Chang, J.-Y. Kim, W.-S. Cho, C.-M. Park, and I.-Y. Song, "Octree-R: An adaptive octree for efficient ray tracing," *IEEE Trans. Vis. Comput. Graphics*, vol. 1, no. 4, pp. 343–349, Dec. 1995.

[45] M. Liu, K. Gu, and J. Qiao, "Convolutional neural network for smoke image super-resolution," in *Proc. 2nd Int. Conf. Comput. Sci. Appl. Eng. (CSAE)*, 2018, pp. 1–6.

[46] K. Bai, W. Li, M. Desbrun, and X. Liu, "Dynamic upsampling of smoke through dictionary-based learning," 2019, *arXiv:1910.09166*. [Online]. Available: http://arxiv.org/abs/1910.09166

[47] M. Werhahn, Y. Xie, M. Chu, and N. Thuerey, "A multi-pass GAN for fluid flow super-resolution," *ACM Comput. Graph. Interact. Techn.*, vol. 2, no. 2, pp. 1–21, Jul. 2019.

[48] R. Bridson, J. Houriham, and M. Nordenstam, "Curl-noise for procedural fluid flow," *ACM Trans. Graph.*, vol. 26, no. 3, p. 46, 2007.

[49] X. Zhang, R. Bridson, and C. Greif, "Restoring the missing vorticity in advection-projection fluid solvers," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–8, Jul. 2015.

[50] F. Christen, B. Kim, V. C. Azevedo, and B. Solenthaler, "Neural smoke stylization with color transfer," 2019, *arXiv:1912.08757*. [Online]. Available: http://arxiv.org/abs/1912.08757

BYEONG-SUN HONG received the B.S. degree in electronics engineering from Chungbuk National University, in 2018, and the M.S. degree in interdisciplinary program in visual information processing from Korea University, in 2020. His current research interests include geometry processing, physically-based simulation, and deep learning.

QIMENG ZHANG received the B.S. degree from the Department of Computer Science and Technology, Zhengzhou University, in 2015, and the M.S. degree in interdisciplinary program in visual information processing from Korea University, in 2018, where she is currently pursuing the Ph.D. degree in interdisciplinary program in visual information processing. Her current research interests include geometry processing, physically-based simulation, virtual reality, and deep learning.

CHANG-HUN KIM received the B.A. degree in economics from Korea University, in 1979, and the Ph.D. degree from the Department of Electronics and Information Science, Tsukuba University, Japan, in 1993. After graduation, he joined the Korea Advanced Institute of Science and Technology (KAIST) as a Research Scientist, where he was involved in many national research projects in the area of computer aided design and geometric modeling. From 1993 to 1995, he was the Head of the Human Interface and Graphics Laboratory, System Engineering Research Institute (SERI). He 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 a member of the IEEE Computer Society and the ACM.

JUNG LEE is currently an Assistant Professor with the Department of Convergence Software, Hallym University. His current research interests include augmented/virtual reality, fluid animation, and computer graphics.

JONG-HYUN KIM received the B.A. degree from the Department of Digital Contents, Sejong University, in 2008, and the M.S. and Ph.D. degrees from the Department of Computer Science and Engineering, Korea University, in 2010 and 2016, respectively. He is currently an Associate Professor with the School of Software Application in Kangnam University. His research interests include physics-based simulation, natural phenomenon modeling, and virtual production.