Journal for Geometry and Graphics
Volume 22 (2018), No. 1, 115–127.

# Data-Driven Approach for Enhancing Splashing Effects of Liquid Simulations

**Takashi Kanai, Mengyuan Wan, Fumiko Enomoto**

*The University of Tokyo, Graduate School of Arts and Sciences*
*3-8-1 Komaba, Meguro-ku, Tokyo 153-8902, Japan*
*email:* kanait@acm.org

**Abstract.** In liquid simulation, splashing is a phenomenon where small particles scatter in the movement of liquids. Fine liquid simulation on the high-resolution grid is indispensable for obtaining detailed splashing effects. However, simulation costs increase and become too high to bear with increasing resolution. Various data-driven methods based on the idea of reusing pre-computed data have been attempted to shorten simulation time, but cannot be easily applied to liquid simulation directly.

In this paper, we propose a data-driven method for enhancing visual effects such as splashing in liquid simulation. Our method successfully applies the data-driven approach to the FLIP method which is suitable for liquid simulation. Moreover, velocity fields can be up-sampled directly in run-time simulation. This enables enhancement of the splashing effect with the change in the topology of fluid surfaces. Our method is able to apply liquid motions that are different from pre-computed scenes. This realizes more flexible enhancements in run-time simulation. We also discuss database creation and usage, especially whether multiple databases of vector fields improve the quality of enhancement or not.

*Key Words:* Data-driven method, Liquid simulation, Splashing, FLIP, SVD

*MSC 2010:* 68U05, 65D15

## 1. Introduction

Due to the growing demands for entertainment and scientific visualization, high-resolution fluid simulation is playing an increasingly significant role in computer graphics research and development. Among different fluid simulation approaches, liquid simulation such as water simulation is a challenging area because of its diversity and complexity. Here we focus on splashing effects in such liquid simulations.

*Splashing* means the scattering of small particles in the collision of liquids. It usually occurs when a liquid moving at a high speed collides into a solid or another liquid. This

ISSN 1433-8157/$ 2.50 

effect can be seen in water scenery like waterfall, dam breaking, and ocean waves. Fine liquid simulation on the high-resolution grid is indispensable for obtaining detailed splashing effects. However, simulation costs increase and become too high to bear with increasing resolution.

For this reason, various data-driven methods based on the idea of reusing pre-computed data have been attempted to shorten the simulation time. However, previously proposed data-driven methods such as the addition of pre-computed turbulence to the fluid surface [7, 6, 10] and model reduction methods [16, 5] are difficult to apply to liquid simulation directly.

In this paper, we propose a data-driven method for enhancing visual effects such as splashing in liquid simulation. Our method successfully applies the data-driven approach to the FLIP method which is suitable for liquid simulation. Moreover, velocity fields can be directly up-sampled in run-time simulation. This enables enhancement of the splashing effect with the change in the topology of fluid surfaces. Our method is able to apply liquid motions that are different from pre-computed scenes. Moreover, our database is constructed from a sequence of small blocks subdivided from the whole simulation region, whereas the whole simulation region itself is required in the case of conventional approaches such as [5]. This realizes more flexible and quicker enhancements in run-time simulation.

Section 2 briefly introduces past studies related to our method, Section 3 discusses details of our proposed method, and Section 4 presents some of the results obtained by our method and a discussion focusing on database creation and usage. Finally, Section 5 outlines the conclusions of our method and prospects for future work.

## 2. Related work

This section specifically discusses splashing liquids and data-driven methods as related work amid the large number of researches which have been conducted on fluid simulation.

### Splashing liquids

There are very few researches focusing on the splashing effects of liquid simulation. Raveendran et al. [12] associated coarse grid with SPH to enforce a divergence free velocity field, allowing their approach to simulate scenarios with significant pressure gradients such as splashing liquids. Thürey et al. [15] and Ueda and Fujishiro [17] focused on the simulation of surface tension in splashing fluids. Yang et al. [19] simulated the spray phenomena of fluids such as fountain and waterfall as well as adopted a hybrid solver of Eulerian grid and Lagrangian particles. Gerszewski and Bargteil [4] conducted the first trial on the simulation of the splashing effect of liquids in a large-scale scenario.

As far as we are concerned, splashing effects are still not fully applied today, especially in large-scale cases. We will thus attempt to develop a novel method focusing on large-scale simulation of splashing phenomena in this paper.

### Data-driven fluid simulation

Since fluid simulation is considerably time-consuming, adaptive methods have been proposed for accelerating fluid simulations [9, 3, 1, 2]. However, it is also true that adaptive methods have certain limits in terms of acceleration.

On the other hand, data-driven approaches by integrating pre-computed noises to low-resolution velocity fields to enhance the details at the post-processing stage have been proposed [7, 6, 10]. Such approaches enjoy the advantage of not needing to solve Navier-Stokes equations on high-resolution grid. These approaches successfully add small-scale details to an

existing flow but fail to naturally add large-scale vortex noises. Moreover, those approaches are not suitable for splashing as they actually do not change the topology of liquid surfaces.

Other types of data-driven approaches using pre-computed data have also been proposed [16, 18, 14, 5]. These approaches are also well known as *subspace methods*. However, they are more like play-back by adjusting parameters than simulation, and are unable to simulate motions that do not appear in the training stage for learning data.

More recently, LADICKÝ et al. [8] proposed a method for predicting liquid motions based on machine learning approach. However, this method requires enormous volume of learning data for making predictions. Thus data size and computational costs required for precompution are serious bottlenecks.

Unlike the above data-driven approaches, instead of applying SVD to the whole simulation region, SATO et al. [13] first divided the simulation region into smaller blocks to gain flexibility in the re-simulation phase. Our method is most inspired by their work and extends to liquid simulation. Our results are thus different from other subspace methods which are not suitable for liquid simulation.

## 3. Data-driven approach for enhancing splash effects

The following describes our data-driven approach for enhancing the splashing effects of FLIP-based liquid simulation. The FLIP (FLuid-Implicit-Particle) method [20] is a hybrid method of Eulerian and Lagrangian methods and is said to come with both advantages. Liquid surfaces are calculated from Lagrangian particles, whereas liquid motions are mainly determined by the vector field on Eulerian grid. Such a grid structure enables application of matrix operations thanks to the structured distribution of velocities which can be easily re-arranged into a matrix form. This prompted us to propose our data-driven approach based on FLIP-based liquid simulation.

We first take a sequence of vector fields in low-resolution liquid simulation as input, and an enhanced corresponding sequence of high-resolution vector fields as output. In our definition,

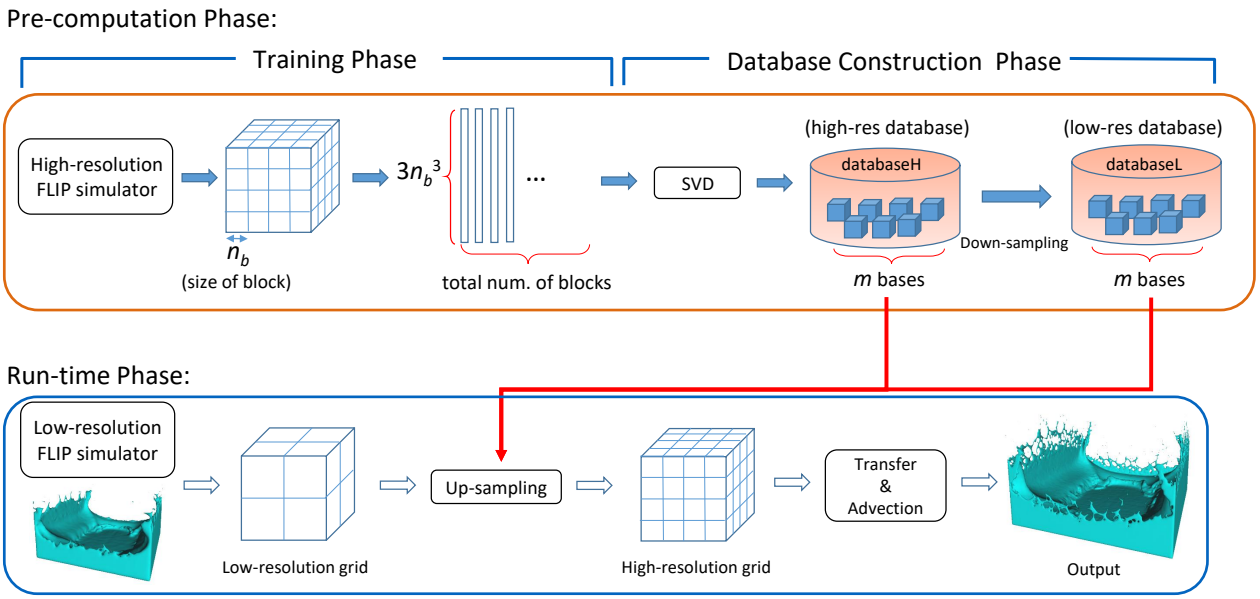


Figure 1: System overview.

**Algorithm 1:** Training Phase

**Input**: None
**Output**: Blocked velocity fields
`// Initialization`
Initialize the MAC grid and insert particles to fluid cells;
`// Simulation`
**for** $n = 0$ *to* $MAX\_STEP$ **do**
  Add external forces to particles;
  Transfer particles' velocities to a grid;
  Save the grid velocities as $\mathbf{v}_{old}$ ;
  Enforce the Dirichlet boundary condition;
  Mark the cells as FLUID, SOLID, or AIR;
  Solve pressure on grid and calculate the new velocity $\mathbf{v}_{new}$ of grid;
  Calculate $\mathbf{v}_{delta} = \mathbf{v}_{new} - \mathbf{v}_{old}$ ;
  Save $\mathbf{v}_{delta}$ by blocks to HDD;
  Calculate velocities for particles;
  Advect particles;

**Algorithm 2:** Database Construction Phase

**Input**: Blocked velocity fields
**Output**: Database databaseL and databaseH
Run out-of-core SVD on blocked velocities;
Select top $k$ eigen-vectors as databaseH;
Down-sample databaseH to databaseL;
Write out databaseH and databaseL;

the number of cells per dimension on the *high-resolution* grid is twice that in the corresponding *low-resolution* grid, thus eight times more in 3D cases. The conversion from low-resolution grid to high-resolution grid is called *up-sampling*. In this case, the up-sampling per step converts each cell to eight smaller cells. This opposite process is called *down-sampling*.

Figure 1 illustrates an overview of our method. Our method mainly comprises two phases; the pre-computation phase and the run-time phase. In the pre-computation phase, a sequence of high-resolution 3D velocity fields are first calculated from FLIP-based liquid simulation. After compressing such vector fields to compute a set of eigen vectors, they are stored in a database. In the run-time phase, a sequence of low-resolution 3D velocity fields are calculated by liquid simulation. Each of these vector fields is up-sampled to its high-resolution vector field by using a pre-computed database. Particles are then advected according to the resulting high-resolution velocity field.

As an additional description of our proposed method outlined in Figure 1, we present the algorithms of the training phase (Algorithm 1), the database construction phase (Algorithm 2) and the run-time simulation phase (Algorithm 3). In Algorithm 1 and Algorithm 3, commands colored in red denote additional processes from the original FLIP method.

**Algorithm 3:** Run-time Simulation and Up-sampling Phase

**Input**: Database databaseL and databaseH
**Output**: Enhanced results
`// Initialization`
Initialize the MAC grid and insert particles to fluid cells;
`// Simulation`
**for** $n = 0$ *to* $MAX\_STEP$ **do**
  Add external forces to particles;
  Transfer particles' velocities to a grid;
  Save the grid velocities as $\mathbf{v}_{old}$ ;
  Enforce the Dirichlet boundary condition;
  Mark the cells as FLUID, SOLID, or AIR;
  Solve pressure on grid and calculate the new velocity $\mathbf{v}_{new}$ of grid;
  Calculate $\mathbf{v}_{delta} = \mathbf{v}_{new} - \mathbf{v}_{old}$ ;
  Up-sample $\mathbf{v}_{delta}$ to high-resolution $\mathbf{v}^h_{delta}$;
  Calculate velocities for particles;
  Advect particles;

### 3.1. Training phase

Unlike Sato et al.'s method [13] in which only 2D simulation is performed in the training phase, we conduct 3D FLIP simulation on a high-resolution grid.

Now there are three types of velocities to be stored in the database; $\mathbf{v}_{old}$, $\mathbf{v}_{new}$ and $\mathbf{v}_{delta} \equiv \mathbf{v}_{new} - \mathbf{v}_{old}$ . Among three vectors, we basically use $\mathbf{v}_{delta}$ for the database. If we use $\mathbf{v}_{new}$ and $\mathbf{v}_{old}$ , both two vectors have to be up-sampled in the run-time phase and more errors are accumulated. Moreover, the PIC velocity calculated from $\mathbf{v}_{new}$ must not be up-sampled because the PIC velocity denotes the viscosity of liquid, meaning that it does not contribute to the splashing.

After $\mathbf{v}_{delta}$ is calculated at each time-step, a grid of $\mathbf{v}_{delta}$ is subdivided into blocks of size $n_b \times n_b \times n_b$, where $n_b$ is the number of cells in each dimension of a block. A set of blocks is stored to HDD block by block for future computation in the database construction phase.

The size of the matrix written to HDD is $m \times n$, where $m = 3n_b^3$ and $n = Ts$, $T$ is the number of frames, and $s$ is the number of blocks on a grid in each frame.

### 3.2. Database construction phase

In the database construction phase, our method applies SVD to a matrix of block velocities stored in the training phase. We then extract the top $k$ principal components of such a matrix, where $k$ is a user-specified parameter but our experiments showed that $k = 128$ components is a good choice. We call these principal components *base velocity*, or BV for short. Afterwards, BVs are stored in the matrix form as the *databaseH*, which stands for *high-resolution database*. We finally down-sample the databaseH to obtain *databaseL*, which stands for *low-resolution database*.

#### Out-of-core SVD

With our method, a 3D velocity field is used to generate BVs. It is often the case that a matrix of such a velocity field may be too large to fit into the main memory. A simple in-core SVD

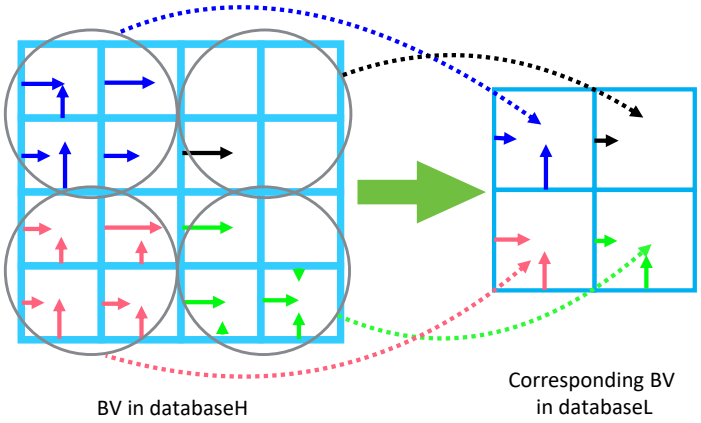


Figure 2: Illustration of down-sampling in case of 2D. All velocities marked with same color are averaged.

is then not applicable when the principal components of a huge matrix are needed. Instead, we use out-of-core SVD [11] in such cases.

**Down-sampling**
To obtain databaseL, we perform a down-sampling step for each BV. Figure 2 illustrates the details of down-sampling operation in 2D cases.

In a down-sampling step, eight sub-blocks in each BV are integrated to a block. In this case, an integrated velocity in each dimension is taken as the average of the corresponding velocities of the eight sub-blocks.

### 3.3. Run-time phase

In the run-time phase, a FLIP-based low-resolution 3D liquid simulation is executed. At each time-step of the simulation, a low-resolution velocity field is converted to a high-resolution velocity field using the BVs stored in the database. Let us denote an input 3D velocity field as $\mathbf{v}_{in}(n_x, n_y, n_z)$ where $n_x \times n_y \times n_z$ is the number of cells in a grid. Our purpose here is to create a up-sampled 3D velocity field $\mathbf{v}_{out}(2n_x, 2n_y, 2n_z)$, meaning that the number of cells in each dimension becomes double.

In the up-sampling process, each block of $\mathbf{v}_{in}$ is separately converted to a high-resolution block of $\mathbf{v}_{out}$. The convertion process is summarized as follows:

- An input block of velocities $\mathbf{v}^l$ is first approximated by the weighted sum of the down-sampled BVs in databaseL.
- The computed weights are then applied to the original BVs in databaseH.
- The output block of velocities $\mathbf{v}^h$ is calculated by the weighted sum of the original BVs.

This process is illustrated in Figure 3.

**Least-squares method**
We accordingly apply the least-squares method to compute the weights $\mathbf{w} = \{w_i\}$ using the BVs in databaseL,

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left( \| \mathbf{v}^l - \sum_{i=0}^{m-1} w_i \mathbf{b}_i^{\downarrow} \|_2^2 \right), \tag{1}$$

where $v^l$ is the input low-resolution block of velocities and $\mathbf{b}_i^{\downarrow}$ the $i$-th column of databaseL.

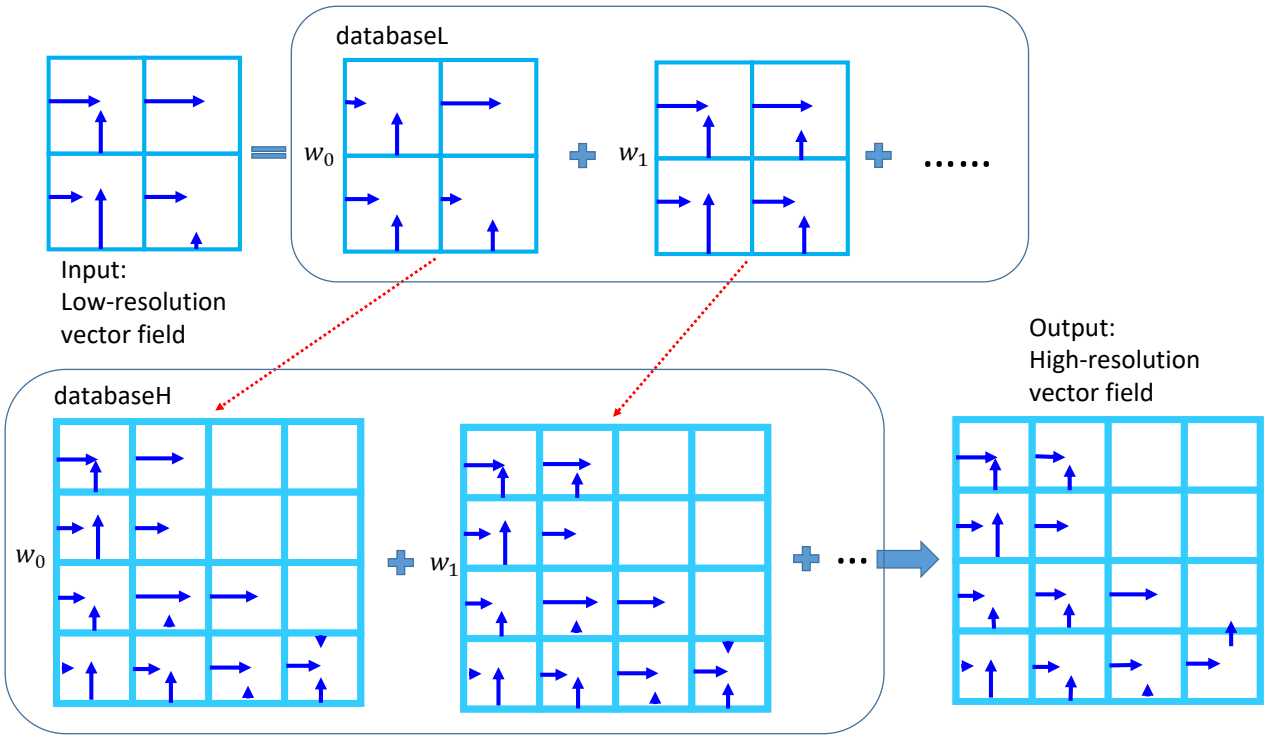


Figure 3: Illustration of up-sampling process in 2D case.

There are several methods to solve the least-squares problem. We adopt here a QR decomposition based approach for the robust computation of optimal weights.

**Adaptive up-sampling**

With the weights $\mathbf{w}$ calculated above, it is possible to directly compute a high-resolution block of velocities $\mathbf{v}^h$ as follows,

$$\mathbf{v}^h = \sum_{i=0}^{m-1} w_i \mathbf{b}_i^{\uparrow}, \tag{2}$$

where $\mathbf{b}_i^{\uparrow}$ is the $i$-th column of databaseH.

However, several problems are met when we apply up-sampling operations to all of blocks. One is the *still water problem*. As there always remains errors during the up-sampling process no matter how high the accuracy is, artifacts appear around the boundary of blocks as shown in Figure 4a. Due to the nature of liquid surface construction, these artifacts can be clearly observed. Moreover, these artifacts cannot be eliminated even when neighbor blocks are overlapped.

Another problem faced in up-sampling is the *boundary problem* around obstacles. Since our training phase does not support the scenes with obstacles and our training data do not include blocks with boundaries, there may exist obvious artifacts around the boundaries of obstacles (Figure 4c). Besides, there is no need to up-sample the blocks in deep water because they are steady and do not contribute to the splashing effect. Therefore, leaving out the up-sampling operation for blocks where obstacles are included or where they are in deep water does not lead to the loss of details and eliminates the need for computation for up-sampling.

To address the above problems, we introduce the adaptive up-sampling. This is based on the idea that only the blocks with low coherence of velocity directions will be up-sampled

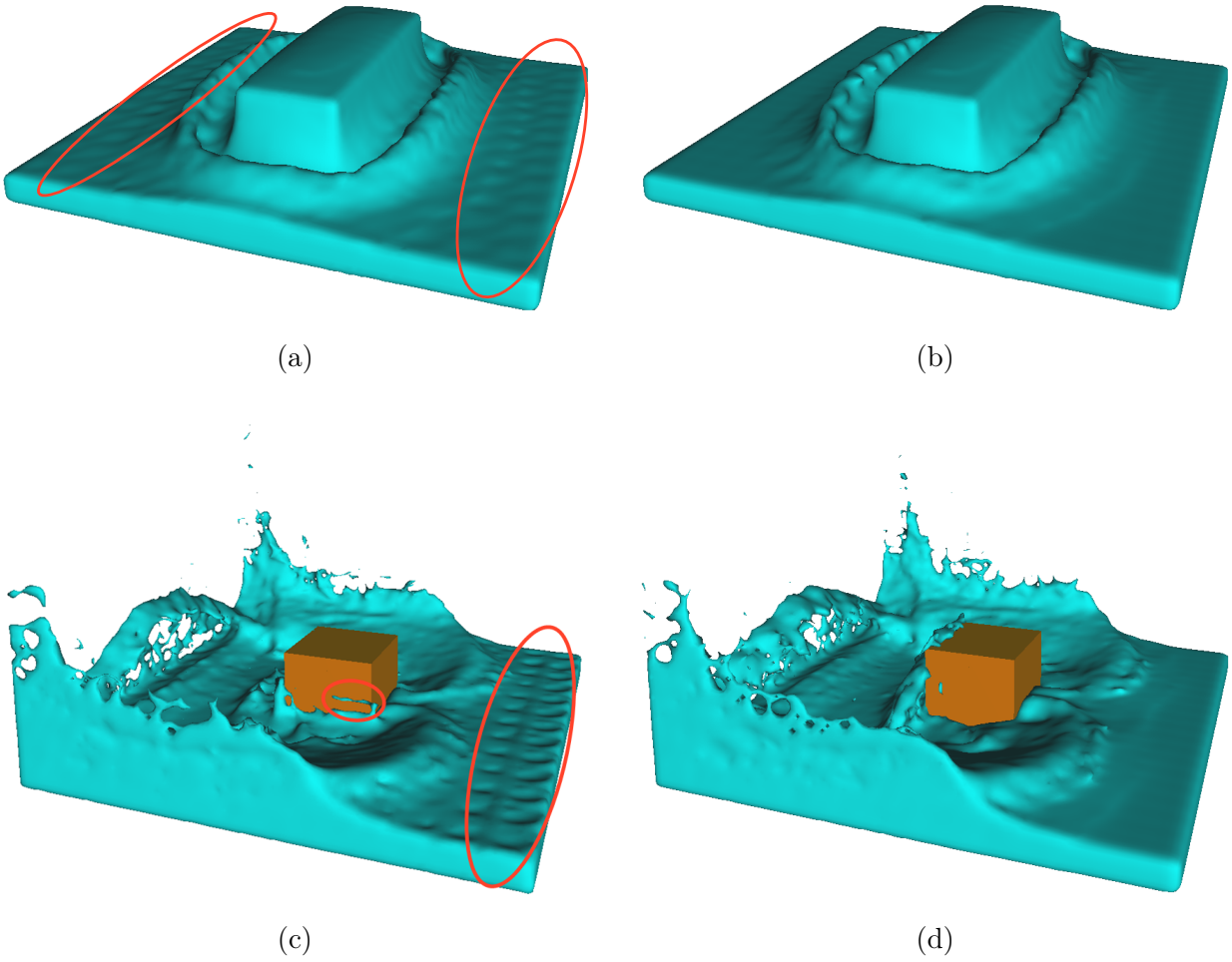

Figure 4: Still water problem and boundary problem. (a) With artifact near still water. (b) Artifact fixed by our adaptive up-sampling method. (c) With artifact near obstacle boundaries as well as still water. (d) Artifact fixed by our adaptive up-sampling method.

(see Figure 5). Specifically, we define the *angle consistency* $\phi$,

$$\phi = \frac{\sum_i^{N_b} |\cos \Delta\theta_i|}{N_b}, \tag{3}$$

where $N_b = n_b^3$ is the number of cells in a block, $\Delta\theta_i$ is the angle between the velocity's direction in a cell $i$, and the average direction of velocities in the corresponding block. The value of $\phi$ is between 0 and 1. When $\phi$ is close to 0, the velocity's direction in a block tends to be different according to the block. Also, when $\phi$ is close to 1, all velocities have the same direction. By adjusting the threshold for $\phi$, we are able to eliminate the still water problem and also control the up-sampling level (see Figures 4b and d).

## 4. Results and discussion

All our experiments were conducted on a desktop PC under Linux OS equipped with Intel® Core™ i7-4770 3.40GHz CPU, nVIDIA® GeForce™ GTX 760 GPU, and 32GB RAM.

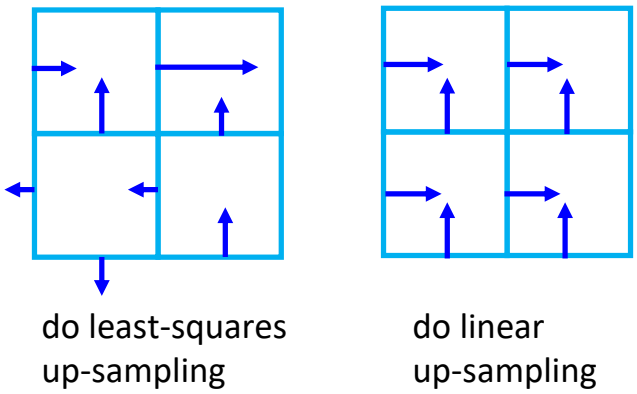


Figure 5: Least-squares up-sampling case (left) and linear up-sampling case (right). Note that velocity field in the right is much more coherent than the left.

Figure 6 shows the comparison between our data-driven results and ground-truth results. In this experiment, the initial scene used in high-resolution simulation and that executed by run-time low-resolution simulation are the same. The top row shows the ground-truth result simulated on the $80^3$ grid. The bottle row shows the result simulated on a $158^3$ grid which is also used for obtaining the learning data. The middle row shows the result of our data-driven method. It can been seen from the comparison result that all our results are enhanced just like the high-resolution ground-truth in terms of the geometry of the liquid surface, number

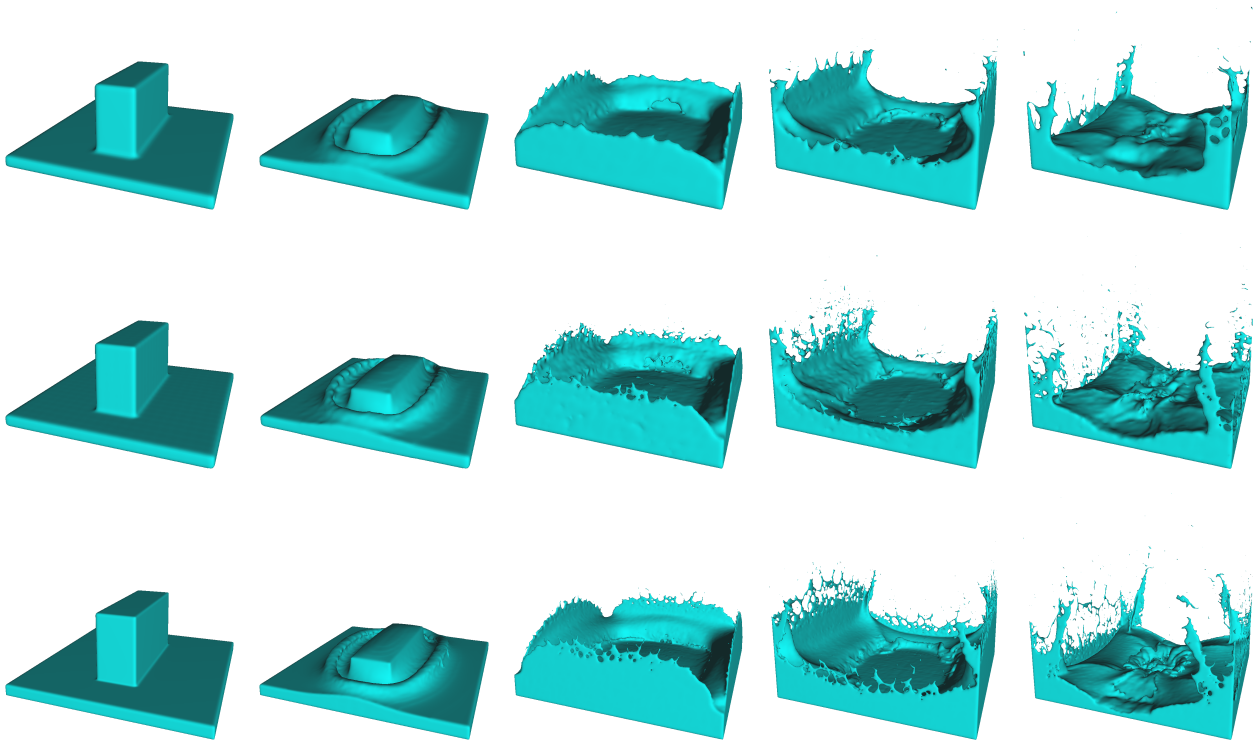

Figure 6: Comparison results. From left to right: 1st, 35th, 70th, 95th, and 130th frame. Top: Ground-truth on $80^3$ grid. Middle: Our data-driven result on $80^3$ grid. Bottom: Ground-truth on $158^3$ grid.

Table 1: Statistical results. From left to right: phase (ground truth (gt) or data-driven (dd) or out-of-core SVD (svd)), grid resolution (#gr), number of particles (#pt), simulation time (minutes), number of frames (#fr).

| *phase* | *#gr* | *#pt* | *time (min.)* | *#fr* |
|---|---|---|---|---|
| gt | $80^3$ | 360,576 | 34 | 300 |
| gt | $158^3$ | 2,955,968 | 643 | 300 |
| svd | $158^3$ | - | 342 | - |
| dd | $80^3$ | 360,576 | 45 | 300 |

of splashing particles, behavior of liquid movement, etc.

Statistical results including the computation time are listed in Table 1. In FLIP-based simulations, the $80^3$ ground-truth simulation for 300 frames takes about 34 minutes, and the $158^3$ ground-truth 643 minutes. Moreover, the computation of BVs by out-of-core SVD takes about 342 minutes. On the other hand, our data-driven method in the run-time simulation takes 45 minutes, which is slightly longer than the $80^3$ ground-truth simulation. Note that although the total pre-computation time adds up to $643 + 342 = 985$ minutes, which seems to be a long time at first glance, the database needs only to be learnt once and can be repeatedly used in run-time simulation.

Figure 7 shows the results of two other enhancements carried out with our data-driven method. The top row is the scence with the different position of a dam, and the bottle row is the scene of using a box as the obstacle. As the learning data, the same database calculated in the experiment shown in Figure 6 is used. The results indicate that the enhancement can be successfully carried out with scenes different from the training data.

**Evaluation about database creation and usage**

With our method above, a sequence of vector fields by the high-resolution simulation of only a scene is compressed and a database is created as basic velocities in the pre-computation phase. However, we believe that the resulting quality differs according to the database creation and

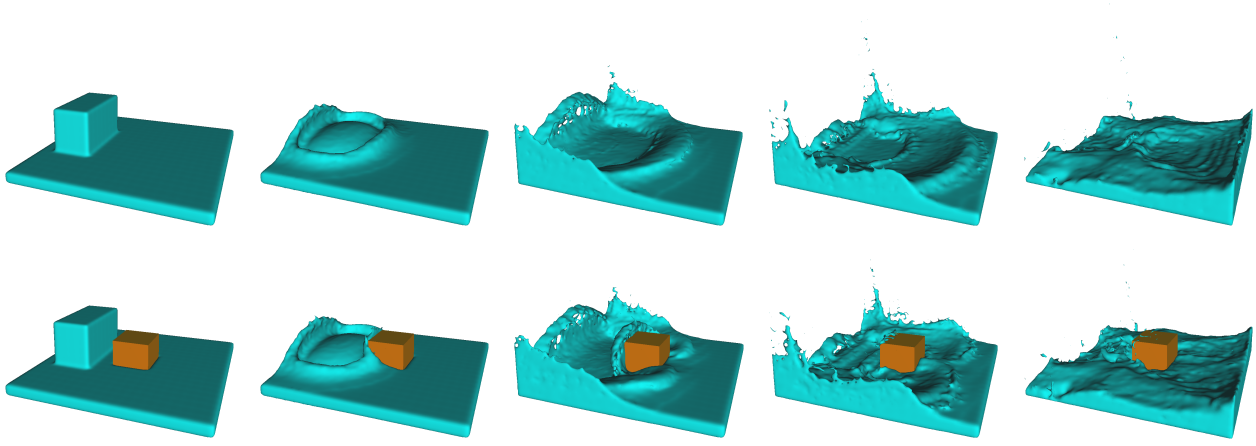

Figure 7: More data-driven results. From top to bottom: 1st, 35th, 70th, 95th, 130th, 180th frame. (a) Data-driven result on $80^3$ grid with initial dam size and position modified. (b) Data-driven result on $80^3$ grid with obstacle.

usage. Furthermore, since there is no need to have a single scene, it may be possible to utilize multiple databases using simulations with several different scenes. We thus evaluated how database creation and usage methods influence enhancement results within the framework of our method.

In the next experiment, databases are created from two different scenes and several up-sampling operations are performed by using such databases. In this experiment, the following four types for database creation and usage are tested:

**MDB-1** Two sequences of vector fields from different scenes are integrated into a single matrix and the BVs of such a matrix is created.

**MDB-2** The BVs of two databases are created independently from two different scenes. In run-time simulation, one of the two BVs with miminum matching error in Equation (1) is selected from the two databases.

**MDB-3** Two sequences of vector fields from different scenes are grouped into two categories according to the angle consistency $\phi$ in Equation (3), and multiple BVs are set per database. In run-time simulation, one of the two BVs with appropriate angle consistency is selected from the two databases.

**MDB-4** Two sequences of vector fields from different scenes are grouped into three categories according to the position on a grid (around the surface, around the boundary, deep water) and the BVs of each category are created. In run-time simulation, one of the three sets of BV with the appropriate position on a grid is selected from the three databases.

We now introduce the *avarage matching error* $E$ as an index to evaluate the up-sampling results. This is an average of the matching errors in Equation (1) taken from all frames and all blocks.

$$E = \frac{\sum^{f}\sum^{b}\left\|\mathbf{v}^{l} - \sum_{i=0}^{m-1} w_i \mathbf{b}_i^{\downarrow}\right\|_2^2}{fb}, \tag{4}$$

where $f$ is the number of frames and $b$ is the number of blocks per frame. This error is used to judge how different the vector field of a low-resolution simulation is from its approximation by the BVs of a database.

Figure 8 shows the experimental results. In run-time simulation, a scene (DB1) is used for low-resolution simulation. Also, in **MDB-1** and **MDB-4**, both two scenes (DB1 and DB2) are used for multiple databases. For reference, the results of using a single database (DB1, DB2) as learning data are shown.

It can be seen from this graph that the up-sampling of **MDB-3** has the smallest average matching error. That is, with our method, better results can be obtained compared with other databases by classifying vector fields with similar features as much as possible. This can also be justified from the viewpoint that the result does not necessarily improve even if a database from a single scene is used, although both scenes in the training phase and in run-time phase are the same.

One issue for **MDB-3** is that the results can be largely changed according to the threshold of angle consistency, and trial and error is required for obtaining the appropriate threshold value. It is necessary to establish a method to automatically determine such parameters in future work.

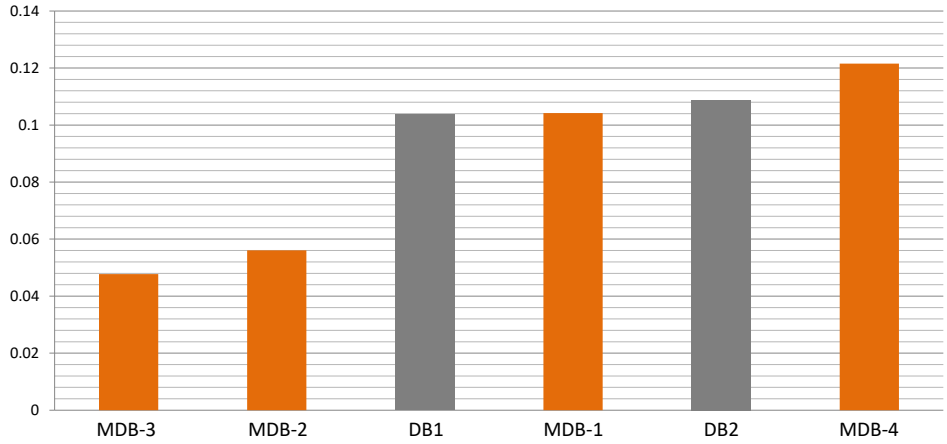


Figure 8: Average matching errors for experiments on database creation and usage.

## 5. Conclusions and future work

We proposed a method to enhance the splashing effect in liquid simulation by using a data-driven approach. With our method, we have demonstrated that the computation time can be largely reduced while keeping the enhancing quality in high-resolution simulation. Through experiments on database creation and usage, we have shown that the use of multiple databases classified according to the angle consistency of vector fields improves the quality of enhancement.

We believe that our method is a promising approach in the area of data-driven fluid simulation. However, the following improvements need to be made in future work.

– First, our method only supports *one-step* up-sampling and we hope to extend it to multi-steps up-sampling to obtain further details while saving more time.

– Secondly, we adopt an adaptive up-sampling method for scenes with obstacles. To improve the accuracy of this method, a database taking into consideration the boundary of obstacles has to be constructed. Moreover, deeper insight into the creation and use of multiple databases is necessary.

– Lastly, we use SVD to obtain the databases. However, there are other algorithms like sparse coding which may improve the enhancement quality of our data-driven method.

## Acknowledgments

We would like to thank Prof. Yoshinori Dobashi of Hokkaido University and Dr. Shuhei Sato of UEI Research for their valuable comments and suggestions.

## References

[1] B. Adams, M. Pauly, R. Keiser, L.J. Guibas: *Adaptively sampled particle fluids.* ACM Trans. Graph. **26**(3), (July 2007).

[2] R. Ando, N. Thürey, C. Wojtan: *Highly adaptive liquid simulations on tetrahedral meshes.* ACM Trans. Graph. **32**(4), 103:1–103:10 (July 2013).

[3] N. Chentanez, M. Müller-Fischer: *Real-time eulerian water simulation using a restricted tall cell grid.* ACM Trans. Graph. **30**(4), 82:1–82:10 (July 2011).

[4] D. Gerszewski, A.W. Bargteil: *Physics-based animation of large-scale splashing liquids*. ACM Trans. Graph. **32**(6), 185:1–185:6 (Nov. 2013).

[5] T. Kim, J. Delaney: *Subspace fluid re-simulation*. ACM Trans. Graph. **32**(4), 62:1–62:9 (July 2013).

[6] T. Kim, J. Tessendorf, N. Thürey: *Closest point turbulence for liquid surfaces*. ACM Trans. Graph. **32**(2), 15:1–15:13 (Apr. 2013).

[7] T. Kim, N. Thürey, D. James, M. Gross: *Wavelet turbulence for fluid simulation*. ACM Trans. Graph. **27**(3), 50:1–50:6 (Aug. 2008).

[8] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, M. Gross: *Data-driven fluid simulations using regression forests*. ACM Trans. Graph. **34**(6), 199:1–199:9 (Oct. 2015).

[9] F. Losasso, F. Gibou, R. Fedkiw: *Simulating water and smoke with an octree data structure*. ACM Trans. Graph. **23**(3), 457–462 (Aug. 2004).

[10] O. Mercier, C. Beauchemin, N. Thürey, T. Kim, D. Nowrouzezahrai: *Surface turbulence for particle-based liquid simulations*. ACM Trans. Graph. **34**(6), 202:1–202:10 (Oct. 2015).

[11] E. Rabani, S. Toledo: *Out-of-core SVD and QR decompositions*. Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing, 2001.

[12] K. Raveendran, C. Wojtan, G. Turk: *Hybrid smoothed particle hydrodynamics*. Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '11, ACM, New York 2011, pp. 33–42.

[13] S. Sato, T. Morita, Y. Dobashi, T. Yamamoto: *A data-driven approach for synthesizing high-resolution animation of fire*. Proceedings of the Digital Production Symposium, DigiPro '12, ACM, New York 2012, pp. 37–42.

[14] M. Stanton, Y. Sheng, M. Wicke, F. Perazzi, A. Yuen, S. Narasimhan, A. Treuille: *Non-polynomial galerkin projection on deforming meshes*. ACM Trans. Graph. **32**(4), 86:1–86:14 (July 2013).

[15] N. Thürey, C. Wojtan, M. Gross, G. Turk: *A multiscale approach to mesh-based surface tension flows*. ACM Trans. Graph. **29**(4), 48:1–48:10 (July 2010).

[16] A. Treuille, A. Lewis, Z. Popović: *Model reduction for real-time fluids*. ACM Trans. Graph. **25**(3), 826–834 (July 2006).

[17] K. Ueda, I. Fujishiro: *Splashing liquids with ambient gas pressure*. SIGGRAPH Asia 2014 Technical Briefs, SA '14, ACM, New York 2014, pp. 6:1–6:4.

[18] M. Wicke, M. Stanton, A. Treuille: *Modular bases for fluid dynamics*. ACM Trans. Graph. **28**(3), 39:1–39:8 (July 2009).

[19] L. Yang, S. Li, A. Hao, H. Qin: *Hybrid particle-grid modeling for multi-scale droplet/spray simulation*. Computer Graphics Forum **33**(7), 199–208 (2014).

[20] Y. Zhu, R. Bridson: *Animating sand as a fluid*. ACM Trans. Graph. **24**(3), 965–972 (July 2005).

Received January 30, 2018; final form February 8, 2018