Interactive Liquid Splash Modeling by User Sketches

GUOWEI YAN, The Ohio State University, USA
ZHILI CHEN, ByteDance, USA
JIMEI YANG, Adobe Research, USA
HUAMIN WANG, The Ohio State University, USA

(a) The sketches drawn in a virtual reality environment (b) The front view of the model (c) The side view of the model

Fig. 1. An artistic liquid splash model in the butterfly shape generated by our interactive modeling system. Given captured user sketches in (a) as input, our system runs a cGAN-based synthesizer to infer a volumetric splash model as output and applies model refinement processes to further improve the model quality. The whole modeling process is straightforward, intuitive and typically takes only a few minutes even for quality results.

Splashing is one of the most fascinating liquid phenomena in the real world and it is favored by artists to create stunning visual effects, both statically and dynamically. Unfortunately, the generation of complex and specialized liquid splashes is a challenging task and often requires considerable time and effort. In this paper, we present a novel system that synthesizes realistic liquid splashes from simple user sketch input. Our system adopts a conditional generative adversarial network (cGAN) trained with physics-based simulation data to produce raw liquid splash models from input sketches, and then applies model refinement processes to further improve their small-scale details. The system considers not only the trajectory of every user stroke, but also its speed, which makes the splash model simulation-ready with its underlying 3D flow. Compared with simulation-based modeling techniques through trials and errors, our system offers flexibility, convenience and intuition in liquid splash design and editing. We evaluate the usability and the efficiency of our system in an immersive virtual reality environment. Thanks to this system, an amateur user can now generate a variety of realistic liquid splashes in just a few minutes.

CCS Concepts: • **Computing methodologies** → **Shape modeling**;

Additional Key Words and Phrases: Sketch-based modeling, liquid modeling, deep learning, generative adversarial network, virtual reality

ACM Reference Format:
Guowei Yan, Zhili Chen, Jimei Yang, and Huamin Wang. 2020. Interactive Liquid Splash Modeling by User Sketches. *ACM Trans. Graph.* 39, 6, Article 165 (December 2020), 13 pages. https://doi.org/10.1145/3414685.3417832

Authors' addresses: Guowei Yan, The Ohio State University, USA, ygwei05@gmail.com; Zhili Chen, ByteDance, USA, iamchenzhili@gmail.com; Jimei Yang, Adobe Research, USA, jimyang@adobe.com; Huamin Wang, The Ohio State University, USA, whmin@cse.ohio-state.edu.

0730-0301/2020/12-ART165 $15.00
https://doi.org/10.1145/3414685.3417832

1 INTRODUCTION

Liquid splashing, caused by the impact of a liquid flow on a solid or liquid surface, is a common and fascinating natural phenomenon in the real world. Artists love to use liquid splashes to create exquisite images and artworks for various purposes and applications. But acquiring liquid splash photographs from the real world is not a simple process, as it requires a photographer to possess specialized equipment and perform considerable practice [Generico 2017]. On the other hand, the complex and rich details of real-world liquid splashes, such as irregular droplets, rupturing splash fronts and capillary waves, make them challenging to model by hand in 3D. Artists can easily spend hours, if not days, on making their models aesthetically appealing and physically plausible, and they cannot conveniently reuse their existing splash models for varying environment setups.

Since the birth of physics-based fluid simulation, computer graphics researchers have been investigating the use of physics-based simulation techniques to model complex liquid phenomena, including liquid splashing. While they have obtained a series of successes in the simulation of thin liquid features [Da et al. 2016; Gao et al. 2018; Wojtan et al. 2010], their techniques are still computationally expensive and not ready for interactive design. What makes the

problem even more challenging is: *how to achieve splash shapes with specific effects*, such as those in Fig. 1 and 13. Due to the complexity of fluid dynamics, setting up the initial simulation condition by hand for specific effects is tedious and time consuming, as it can take a lot of trials and errors. In the past, researchers tried to tackle this problem by solving space-time optimization [Pan and Manocha 2017; Thürey et al. 2009; Treuille et al. 2003], under the assumption that sufficient target shape or flow information has been given. Their solutions are more suitable for directable fluid animation than liquid shape modeling. After all, if our goal is to model static liquid shapes only, why cannot we just do it right away and what is the point of running simulation from scratch?

In this paper, we focus our research on the development of a novel system that allows users, including both amateurs and professionals, to conveniently and interactively create their 3D liquid splash models from simple sketches, without addressing every single shape detail. We are specifically interested in using depth-augmented devices, e.g., VR handheld controllers, to capture user strokes. Compared with other ways of acquiring strokes, depth-augmented devices have a unique advantage in obtaining the 3D trajectory and the 3D velocity at the same time, the latter of which is important to liquid shape control as shown later in Subsection 4.3. As those devices become more accurate and accessible in the future, we believe that our system can be ready for its potential applications. To efficiently synthesize splash shape models from user input, we employ conditional generative adversarial networks (cGANs) [Mirza and Osindero 2014], which have demonstrated their powers in synthesizing images [Isola et al. 2017], terrains [Guérin et al. 2017], super-resolution fluid flows [Xie et al. 2018] and many more. In our system, the input to the generator network, i.e., the condition, is a set of user strokes, and the output is a volumetric liquid splash shape model coupled with its underlying flow. Toward the development of this system, we solve a series of technical problems and we summarize our contributions as follows.

- *Data representation and preparation.* Different from previous sketch-based modeling techniques [Delanoy et al. 2018; Li et al. 2017, 2018] that define sketches as silhouettes and curvature lines on surfaces, our system defines sketches as underlying streamlines, i.e., the paths traced out by immersed particles moving within the flow. Based on this representation, we present our data preparation process that extracts streamlines as user strokes from simulated volumetric fluid data suitable for training.
- *The data set and the synthesizer.* Using physics-based fluid simulation, we build a large data set that contains more than 10,000 liquid splash models with extracted streamlines. We use this data set to train our cGAN-based synthesizer and we demonstrate its effectiveness in inferring a wide range of liquid splash shapes.
- *Physics-inspired model refinement.* One challenge in this research is how to generate rich details comparable to those in real-world splashes. To address this challenge, we introduce a physics-inspired model refinement component into the system. It consists of a particle-based process that enriches models with additional liquid droplets and a mesh-based process that refines models for small-scale capillary waves.

We implement our interactive system and evaluate its usability in an immersive virtual reality environment, using an Oculus Rift S VR gaming headset for display and motion input. Our experiment demonstrates the capability and the efficiency of this system in generating a variety of liquid splash models, including those with desired specific effects as shown in Fig. 1 and 13. Since our splash model contains a velocity field representing the underlying flow, we can use the model immediately for further refinement and animation production without resorting to additional tools for velocity initialization, as shown in Fig. 16 and 17. This feature is of particular importance to future liquid splash applications, which demand finished high-quality results.

2 RELATED WORK

2.0.1 Sketch-based 3D modeling. The idea of using 2D or 3D sketches to facilitate 3D modeling has a long history in computer graphics research. The early work by Cohen et al. [1999] used the sketched 2D projection and shadow of a curve to determine its shape in 3D. Igarashi et al. [1999] studied the construction of 3D polygonal surfaces from 2D silhouette drawings. Nealen et al. [2007; 2005] proposed to draw sketches on mesh surfaces as control handlers for meshing editing. Schmidt and Singh [2008] adopted a tree data structure for non-destructive surface editing by sketches. Lee and Funkhouser [2008] used sketches to search and composite 3D models. Gingold et al. [2009] combined 2D sketches with additional annotations for resolving ambiguities in 2D-to-3D mapping. Rivers et al. [2010] used multi-view silhouettes to create CSG models.

In recent years, researchers have been actively exploring the use of machine learning techniques in sketch-based modeling. Xu et al. [2013] treated sketches as a tool for retrieving and modeling common indoor objects in 3D. Huang et al. [2016] applied neural networks to build a mapping from sketches to the space of parameterized 3D models. De Paoli and Singh [2015] proposed to build layered 3D models by 2D sketches. Guérin et al. [2017] studied the use of conditional generative adversarial networks (cGANs) in sketch-based terrain modeling. Delanoy et al. [2018] used convolutional neural networks (CNNs) to predict the occupancy of a voxel grid from sketches. Li et al. [2018] suggested the use of an intermediate CNN layer and additional add-on sketches for effective shape modeling.

2.0.2 Directable fluid simulation. How to control physics-based fluid simulation for achieving desired effects is an interesting yet challenging problem. Foster and Metaxas [1996] applied varying pressure controllers to control fluid animation. Shi and Yu [2005], Nielsen and Bridson [2011], and Raveendran et al. [2012] investigated how to guide fluid simulation by pre-defined target shapes. Rasmussen et al. [2004] and Thürey et al. [2009] proposed to modify fluid velocities by control particles. In general, fluid control by key frames can be formulated and solved as a space-time optimization problem [McNamara et al. 2004; Pan and Manocha 2017; Treuille et al. 2003], which requires considerable computational costs. Huang et al. [2011] and Nielsen and Bridson [2011] pushed the problem

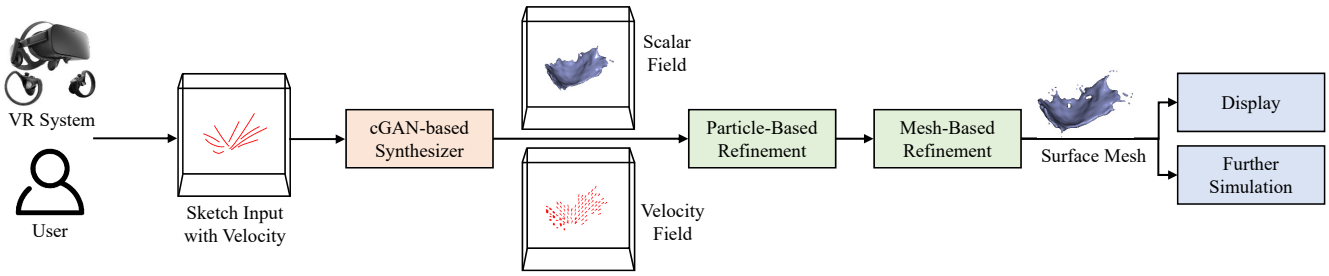


Fig. 2. The system workflow. Our system takes user sketches as input and runs a model synthesis component (in orange) and a model refinement component (in green) to generate 3D liquid splash models.

further, aiming at matching high-resolution simulation with low-resolution preview simulation. In the past, researchers have also investigated fluid simulation control by post-editing [Manteaux et al. 2016; Pan et al. 2013], or the composition of simulated sequences [Raveendran et al. 2014]. Recently, Zhu et al. [2011] and Hu et al. [2019] studied 2D fluid animation by sketches.

2.0.3 Data-driven fluid simulation. Our research is also related to data-driven fluid simulation techniques, especially those based on machine learning. Ladickỳ et al. [2015] proposed to use regression forests to predict particle movements within a large time step, for fast particle-based fluid simulation. Chu and Thürey [2017] trained a CNN for local flow feature descriptors and used it for synthesizing details in smoke simulation, by matching and transferring pre-computed space-time regions. Tompson et al. [2017] also trained a CNN, but treated it as a faster linear solver for pressure projection in Eulerian fluid simulation. Kim et al. [2019] trained a CNN as a generative model for interpolation and time integration of fluid simulation in the latent space, after parameterizing the velocity field. Umentani and Bickel [2018] took a data-driven approach to estimate fluid flows around obstacles for interactive aerodynamic design. Wiewel et al. [2019] trained LSTM networks to predict pressure field changes over time.

Our research is particularly related to the recent work by Xie, Um, Thürey and their collaborators. They used cGANs to enrich a single frame of coarse smoke simulation [Xie et al. 2018], and more recently, learned a neural network model for adding splash droplets into existing liquid simulation [Um et al. 2018]. Our work also uses cGANs and synthesizes splash details, but without coarse simulation.

3 OVERVIEW

Fig. 2 illustrates the workflow of our interactive liquid splash modeling system. The system consists of two major components: a model synthesis component by neural networks and a model refinement component for detail enrichment. Before we discuss them in Section 4 and 5, we will briefly examine the system in this section.

To begin with, the system asks a user to draw through a depth-augmented input device, such as hand tracking glove or VR controller. The device tracks every user stroke and stores it as a polyline. The system then voxelizes all of the input strokes onto a uniform grid, covering the sketch domain around the user. Given the voxelized sketch data as input, our trained cGAN-based synthesizer produces two fields as output: a scalar field representing the probabilistic occupancy of liquid volume, and a velocity field representing the underlying liquid flow. The system can then apply the marching cube method to reconstruct a surface mesh from the scalar field, or treat the two fields as the initial condition for further editing and simulation. Thanks to the efficiency of our model synthesis component, the user can edit the sketches and check the result interactively, until he/she becomes satisfied.

One issue associated with the synthesis component is that it can miss fine details, as shown in Fig. 10a and 12a. We cannot address this issue by simply using higher grid resolution, since that would significantly increase memory and computational costs. Instead we strengthen our system by a model refinement component, which includes a particle-based process for adding extra droplets (in Section 5.1) and a mesh-based process for producing capillary surface waves (in Section 5.2). Fig. 10 to 13 demonstrate the effect of this model refinement component.

4 SPLASH MODEL SYNTHESIS

In this section, we would like to discuss the sketch-based liquid splash model synthesis component based on deep neural networks. We first present the streamline representation of a liquid splash model that simulates user sketches in Subsection 4.1. In Subsection 4.2, we discuss the process of data generation and preparation for training our neural networks. Finally, in Subsection 4.3, we describe the cGAN-based synthesizer that takes voxelized sketch input to generate volumetric occupancy and velocity output.

4.1 Streamline Representation

The first and foremost question we have to answer is: *what do sketches mean with respect to liquid splash models*? Existing sketch-based shape modeling approaches [Delanoy et al. 2018; Li et al. 2017, 2018] typically define sketches as feature and contour curves on surfaces. While this representation is suitable for modeling solids, we found it to be problematic for modeling liquid splashes. The key reason is because liquid splashes are often in complex shapes and it would be impractical and non-intuitive for the user to outline surface shapes as sketches. To solve this problem, we propose not to consider liquid splashes merely as shapes, but as the outcomes of fluid flows. Specifically, we treat user sketches as representative streamlines, which are widely used as descriptors in flow visualization [Lim and Smits 2000]. As the user sketches, the device tracks his/her

movement and records every stroke as a polyline, defined by a number of connected sample points. At every sample point, we store both its position and its velocity, the latter of which is controlled by the user's movement speed. In this way, the user sketches the streamlines of an underlying fluid flow, while the system finishes a liquid splash model as discussed next.

To represent different liquid splashes, we need different numbers of polylines, made of different numbers of sample points. This causes inconsistency within the polyline-based streamline representation for training and synthesis. To solve this issue, we convert the polyline-based representation into the volumetric representation. Given a uniform 3D grid covering the sketch domain around the user, we voxelize every sketched polyline onto the grid. The voxelized result contains two fields: a scalar field representing the binary occupancy of the polylines within each grid cell; and a velocity field representing the streamline velocities. We treat such voxelized sketch data as input to our synthesizer for both training and synthesis next, as shown in Fig. 6.

4.1.1 The mapping from strokes to splashes. We would like to emphasize that there can be many liquid splash shapes under the same liquid flow. The streamline representation is also not unique, nor sufficient to determine the liquid flow uniquely. Therefore, the mapping from user strokes to liquid splashes is not one-to-one, but many-to-many. This is why we choose to use a conditional generative model as discussed in Subsection 4.3, which treats user strokes, not as a unique descriptor, but as conditional guidance during the model synthesis process.

4.2 Data Acquisition by Physics-Based Simulation

To make our synthesizer capable of generating a wide range of liquid splashes from various input sketches, we need a large training data set with quality and diversity. In this subsection, we will first present the generation of fluid data using physics-based simulation. Then we will describe our data preparation process that converts raw fluid data into a training data set.

4.2.1 Data generation. Since it is impractical to obtain such a large data set from hand-drawn sketches and shapes, we resort to physics-based fluid simulation. Our simulator adopts the Fluid-Implicit-Particle (FLIP) method [Brackbill et al. 1988; Zhu and Bridson 2005], which is a hybrid method that uses both Lagrangian particles and Eulerian grids in simulation. In general, any volumetric fluid simulator should be capable of serving the data generation purpose here.

In every simulation scene, we eject a short liquid streamlet toward a solid obstacle and we store the simulated frames 0.05 to 0.5 seconds after the impact, during which most of the interesting splashes appear as our experiment shows. To improve the diversity of our data set, we allow a large number of initial conditions to be random variables, including the length (from 0.05m to 0.25m), the radius (from 0.005m to 0.050m) and the initial velocity (from 0.5m/s to 5.0m/s) of the streamlet, the shape and the size of the solid (selected from sphere, cube, cylinder and cone), and the impact location parameterized by the displacement between the streamlet and the solid. We choose not to include randomized liquid-liquid impact events[1] into our scenes, because most of their splashes are short-lived, dispersed, and can be represented by those in the existing scenes already.

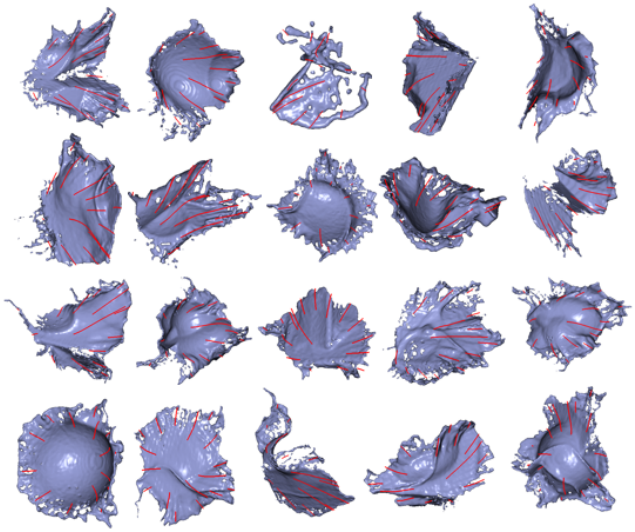


Fig. 3. A subset of liquid splash shapes in our data set. Our system uses physics-based fluid simulation to generate those shapes, by ejecting short liquid streamlets toward solid obstacles in different shapes. The simulation runs on an Eulerian grid with $256 \times 256 \times 256$ cells.

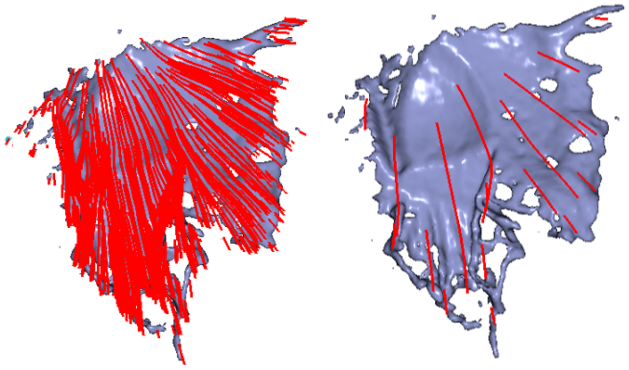
We simulate 1,021 randomized scenes and we set the simulation grid resolution to $256 \times 256 \times 256$, which provides an acceptable balance between the simulation accuracy and the simulation time. In total, it takes 204 hours for our simulator to finish all of the scenes on a single workstation, or 12 minutes per scene. After that, we manually select 8 to 16 frames in every scene as the representative ones for our data set. To facilitate this manual process, we develop a tool that automatically filters out similar frames after every selection. Compared with the raw data set that includes every frame and the sampled data set that includes one frame within a certain time interval, our data set is much more compact yet representative as we found in our experiment. In total, the data set contains 11,843 simulated frames, each of which is a snapshot of the simulation status stored in a uniform grid. Fig. 3 shows a subset of simulated liquid shapes in our data set.

4.2.2 Data preparation. Given the volumetric fluid data generated by our simulator, we must extract their streamlines next as stroke input to our networks for training purposes. By definition, a streamline is the path traced out by a massless particle as it moves within the flow. Let $\mathbf{u}(\mathbf{p})$ be the flow velocity at a 3D location $\mathbf{p}$. The streamline starting from a seed point $\mathbf{p}_0$ is an integral curve:

$$\mathbf{p}(t) = \mathbf{p}_0 + \int_0^t \mathbf{u}(\mathbf{p}(s))ds. \tag{1}$$

Our streamline extraction algorithm uses a uniform seeding strategy [Liu et al. 2006b] to determine the placement of each seed point,

[1] Another reason is because we focus our research on low-viscosity liquids, especially water. Liquid-liquid impact events can be useful in exhibiting the viscous behaviors of high-viscosity liquids, but they are beyond the scope of this work.



(a) Before clustering (b) After clustering

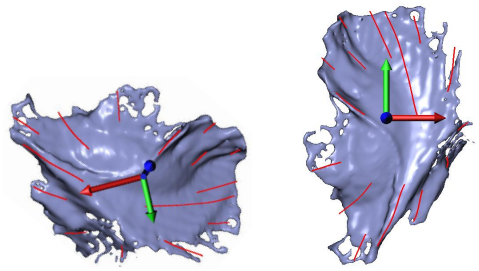
Fig. 4. Streamlines extracted from a simulated splash shape. After we generate densely sampled streamlines as shown in (a), we apply hierarchical clustering to find those representatives ones, as shown in (b). We consider them to be the strokes the user is likely to draw.

and then traces the point forward with a small step size to calculate discrete streamline sample locations, until the streamline reaches a given length, leaves the specified domain, or forms a loop. The result is a set of densely sampled streamlines as shown in Fig. 4a.

However, we cannot treat every calculated streamline as a stroke, because there are too many for the user to draw in reality. To solve this problem, we assume that the user draws those representative ones as shown in Fig. 4b and we apply hierarchical clustering [Rokach and Maimon 2005] to discover them from bottom to up: we assign every streamline with its own cluster initially and we merge two similar clusters into one repetitively, until the remaining number of clusters drops below a certain threshold. We use the Hausdorff distance as the similarity measure between two clusters. Since the number of strokes the user can draw depends on his/her willingness, we treat the threshold on the cluster number as a random variable from 5 to 20, and we generate multiple streamline data from the same simulated volumetric shape. This is a cheap way for us to augment our data without running more simulations.

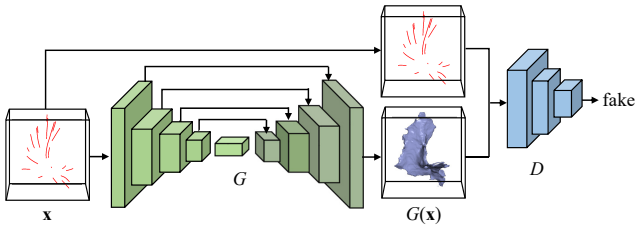
4.2.3 Data alignment. Free-falling splash models are intrinsically invariant to rigid transformation[2] and there is no need for the synthesizer to learn such transformation. Even if it can learn, it would require extensive data augmentation. Because of that, we propose to eliminate rigid transformation in our data through a rigid alignment process. To do so, we first apply principal component analysis on the set of streamlines to calculate its local coordinate system. Using this local coordinate system, we then transform the whole model, including both its streamlines and its volumetric data, from the world space to the local space as shown in Fig. 5. Transforming the volumetric data requires us to resample the grid at the same resolution. Thanks to data alignment, our training process now works in a space invariant to rigid transformation, which is much smaller than before. We note that since our synthesizer is trained for aligned models, we must transform input strokes to the local space and transform the synthesized output back, before and after online modeling synthesis.

[2]Splash models are variant to non-rigid transformation, especially scaling. This is mostly due to surface tension affected by the scale.

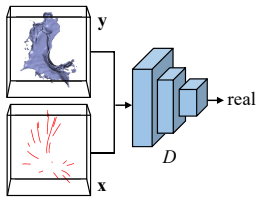


(a) The model in the world space (b) The model in the local space

Fig. 5. A splash model in the world space and the local space. We perform data alignment to convert every splash model from the world space to the local space, so that the aligned model spans a smaller space for training our deep neural networks.



(a) The generator, whose result should be classified by the discriminator as "fake"



(b) The discriminator

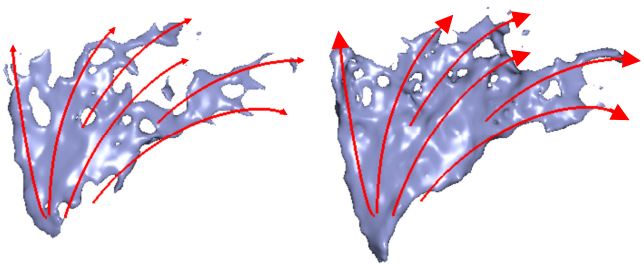
Fig. 6. A cGAN network. In the training process of a cGAN network, the discriminator D is trained to classify synthesized results from simulated examples, while the generator G is trained to fool the discriminator. The discriminator and the generator are trained jointly.

4.3 A cGAN-based Synthesizer

Our synthesizer is based on conditional generative adversarial networks (cGANs) [Isola et al. 2017; Xie et al. 2018]. A cGAN is a conditional generative model that learns a mapping from input $\mathbf{x}$ and random noise vector $\mathbf{z}$ to output $\mathbf{y}$: $\{\mathbf{x}, \mathbf{z}\} \rightarrow \mathbf{y}$. Like regular generative adversarial networks (GANs), a cGAN consists of two network components: a generator G and a discriminator D. Let $\mathcal{L}_{cGAN}(G, D)$ be the objective function of a cGAN:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(\mathbf{x}, \mathbf{y})] + \mathbb{E}_{\mathbf{x},\mathbf{z}}[\log(1 - D(\mathbf{x}, G(\mathbf{x}, \mathbf{z})))], \quad (2)$$

and $\mathcal{L}_{L1}(G) = \lambda_{L1}\mathbb{E}_{\mathbf{x},\mathbf{y},\mathbf{z}}[\|\mathbf{y} - G(\mathbf{x}, \mathbf{z})\|_1]$ be an additional L1 loss term [Isola et al. 2017] for keeping the generator output close to the ground truth. The goal of a cGAN is to find the generator G^* that



(a) Low stroke velocity (b) High stroke velocity

Fig. 7. Synthesized results with different stroke velocities. This figure shows that the stroke velocity is important to splash modeling by our synthesizer. In general, the shape is more ruptured under low stroke velocity as shown in (a), and it remains more like a thin sheet under high stroke velocity as shown in (b). These results are expected, since liquid thin features do not last long in the real world and they are more likely to form at high speed.

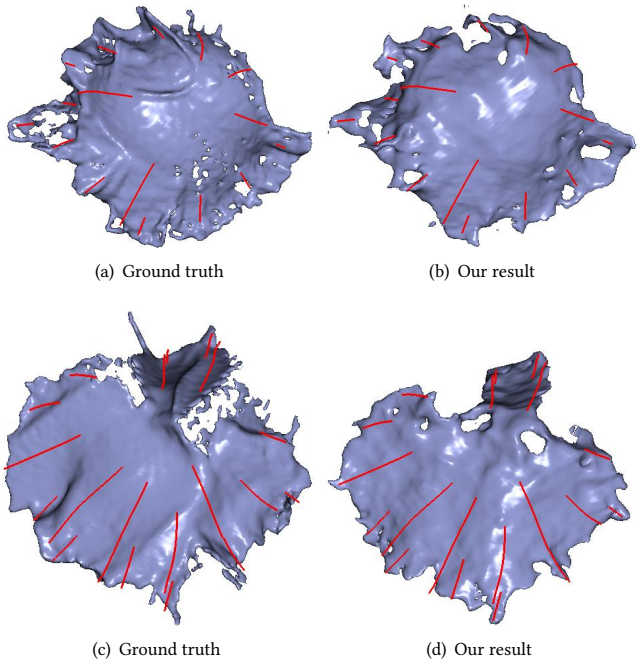
solves the following optimization problem:

$$G^* = \arg\min_G \max_D \left\{ \mathcal{L}_{cGAN}(G, D) + \mathcal{L}_{L1}(G) \right\}. \tag{3}$$

Intuitively, G and D play a two-player minmax game, during which G tries to minimize the overall objective while D tries to maximize it. This game between G and D continues, till they reach an equilibrium state eventually.

Similar to [Isola et al. 2017], we choose U-Net [Ronneberger et al. 2015] but with 3D convolution as the architecture of our generator, as shown in Fig. 6. This generator is an encoder-decoder network. In the encoding stage, a stack of fully convolutional layers progressively reduce spatial resolution and increase feature dimensionality. After that, the generator reverses the process in the decoding stage to recover spatial resolution. The network uses additional skip-connections to link the decoder layer and the encoder layer with the same resolution. Skip-connections help pass low-level input features to output by circumventing the bottleneck of information flow. In our system, we set the filter size as $4 \times 4 \times 4$ and the number of feature channels in the successive layers as 64, 128, 256, 512, 512, 512, and 512. To form the network of our discriminator, we use four convolutional layers with leaky ReLU activation and a fully connected layer for final output.

To implement the synthesizer for generating liquid splash models, we treat the voxelized sketch data in Subsection 4.1 as input $\mathbf{x}$ and the volumetric splash model as output $\mathbf{y}$. Similar to [Isola et al. 2017], we provide the random noise in the form of dropout in training and synthesis processes. In our system, $\mathbf{x}$ and $\mathbf{y}$ are in the same $128 \times 128 \times 128$ grid resolution, both of which contain two fields: a scalar field representing the occupancy and a velocity field representing the underlying flow. Due to the memory limit, this resolution is lower than the resolution of the original simulation data in our data set as discussed in Subsection 4.2.1, which needs down-sampling before use. In total, the dimensionality of $\mathbf{x}$ and $\mathbf{y}$ is $128 \times 128 \times 128 \times 4$. We note that both the stroke trajectory and the stroke velocity are important to liquid splash modeling and they provide the user more controllability. Because of that, we should generate the shape and the flow jointly using a single synthesizer, rather than two separate ones. Fig. 7 compares our splash results when the synthesizer takes the same user stokes but with different velocities as inputs.



(a) Ground truth (b) Our result

(c) Ground truth (d) Our result

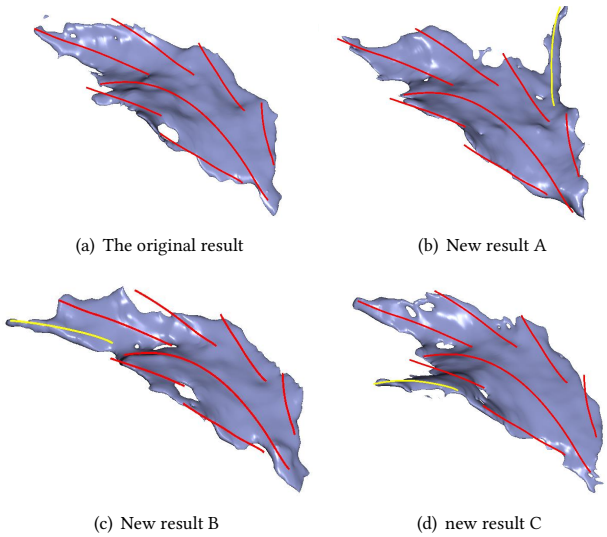
Fig. 8. Our synthesized results in comparison with the ground truths generated by simulation. This figure shows that our synthesizer has sufficient accuracy in predicting the liquid splash shapes from the strokes.

Table 1. Quantitative evaluation of our system by four accuracy metrics. This table also evaluates our system with different λ_{L1} values, in comparison with a baseline U-Net model.

Model	IoU	TPR	FPR	Hausdorff
Ours ($\lambda_{L1} = 10$)	0.628	0.721	0.0031	21.5mm
Ours ($\lambda_{L1} = 100$)	0.735	0.816	0.0013	9.5mm
Ours ($\lambda_{L1} = 1000$)	0.713	0.791	0.0017	10.5mm
U-Net	0.685	0.768	0.0026	16.0mm
Ours (after refinement)	0.733	0.817	0.0015	9.6mm

4.3.1 Training networks. To train the networks, we use the TensorFlow framework and the Adam optimizer [Kingma and Ba 2015] with a learning rate of 10^{-4}. We set the batch size to two, which is the highest we can get within the memory limit. Our training process uses batch normalization [Ioffe and Szegedy 2015] and dropout [Srivastava et al. 2014] for regularization. In total, it takes 235 hours to finish 120 epochs on a single workstation.

4.3.2 Accuracy analysis. Our test data set includes 3,261 frames selected from 307 newly simulated scenes. We use our synthesizer to model liquid splash shapes from extracted strokes and compare them with simulation results as ground truths. In general, our synthesizer can predict the overall liquid shapes reasonably well, although it tends to lose some fine details as shown in Fig. 8. Table 1 provides



(a) The original result (b) New result A

(c) New result B (d) new result C

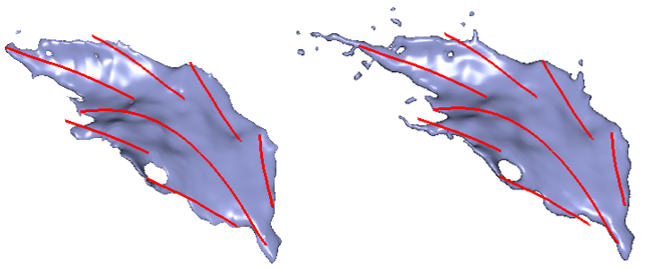
Fig. 9. Our new results synthesized after adding new strokes (in yellow), in comparison with the original result of using existing strokes (in red). This figure indicates that our results are coherent through the sketching process.

quantitative evaluation of our system by four accuracy metrics: the intersection over union (IoU) on the occupancy grid, and the true positive rate (TPR) and the false positive rate (FPR) of each voxel, and the Hausdorff distance between our result and the ground truth. In addition, Table 1 evaluates our system with different λ_{L1} values, in comparison with a baseline U-Net model. It indicates that a suitable L1 loss term is important to the accuracy of our system, while our system outperforms the base line model most of the time.

4.3.3 Sensitivity analysis. To evaluate the sensitivity of our synthesizer with respect to stroke input, we design another experiment that compares our synthesized results before and after adding new strokes (in yellow), as shown in Fig. 9. Since our synthesizer takes all of the strokes together as input, adding a new stroke causes the whole liquid splash shape to change as expected. Fortunately, our results still exhibit sufficient overall coherence through the sketching process, which is important for the user to perform sketch-based design in a controllable way. If the user really wants to keep a new stroke from affecting other splash parts, a natural solution is to design the model in pieces. For example, each butterfly wing in Fig. 1 is created as a single liquid splash.

5 SPLASH MODEL REFINEMENT

Real-world liquid surfaces often exhibit complex small-scale details. Unfortunately, our synthesizer alone has difficulty in producing rich small-scale details for two reasons. First, the training of our synthesizer uses physics-based simulation data, which do not contain as many details as real-world splashes do due to their intrinsic limitations [Gao et al. 2018; Um et al. 2018; Wojtan et al. 2010]. Second, the synthesizer can cause further detail losses as shown in Subsection 4.3.2, due to coarse grid resolution and imperfect neural networks. Motivated by recent research on enriching liquid animation with particles [Ihmsen et al. 2012; Roy et al. 2020] and



(a) Before refinement (b) After refinement

Fig. 10. The surface mesh results before and after running the particle-based refinement process. This process uses liquid and air particles to improve surface details near splash fronts, as shown in (b).

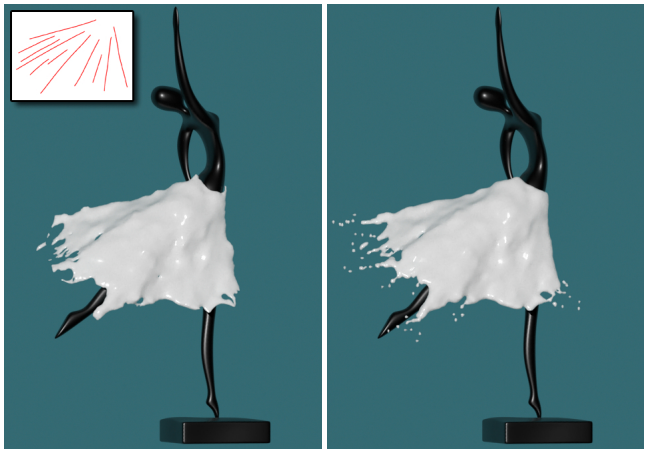
waves [Canabal et al. 2016; Jeschke and Wojtan 2017; Kim et al. 2013; Mercier et al. 2015], we would like to enrich synthesized liquid splashes with small-scale details as well. A unique challenge we face here is a lack of dynamic information, since we are dealing with static shapes, not animation sequences. Therefore we propose to formulate our model refinement processes in a physics-inspired fashion.

5.1 Particle-Based Refinement

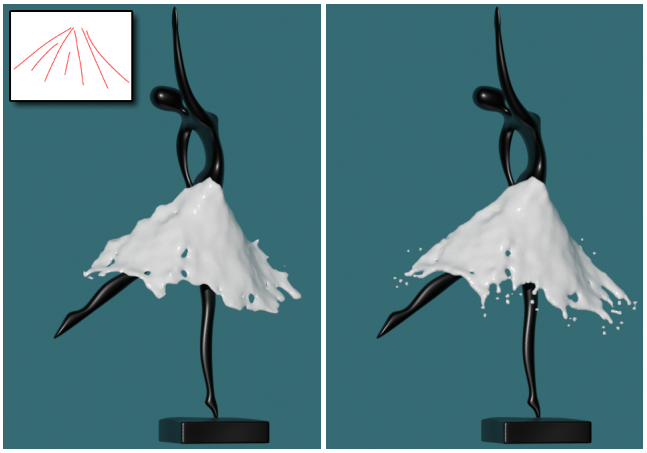
The key idea behind our particle-based refinement process is to use particle-based fluid simulation to enrich or trim the synthesized liquid splash model. Let $\phi(\mathbf{q})$ be the scalar field and $\mathbf{u}(\mathbf{q})$ be the velocity field produced by our synthesizer in Section 4. We first apply stratified random sampling to select a set of sample points $\{\mathbf{q}_i\}$, such that every point $\mathbf{q}_i$ satisfies:

$$\phi_{\text{air}} \le \phi(\mathbf{q}_i) < \phi_{\text{air}}+\phi_{\text{band}}, \quad \left\|\nabla^2\mathbf{u}(\mathbf{q}_i)\right\|^{\alpha} \cdot \left\|\nabla \cdot \left(\frac{\nabla\phi(\mathbf{q}_i)}{\|\nabla\phi(\mathbf{q}_i)\|}\right)\right\|^{\beta} \ge \gamma, \tag{4}$$

where ϕ_{air} is the scalar threshold specifying the air-liquid interface, ϕ_{band} is the interface sampling bandwidth, α and β are the exponential variables, and γ is the sample selection threshold. Essentially, the second part of Eq. 4 provides the metric for choosing sample points in turbulent and curved regions, where surface ruptures are likely to occur and liquid droplets are likely to appear. We use finite differencing [Osher and Fedkiw 2003] to evaluate Eq. 4. We then generate 100 to 400 blue noise particles around every sample point by the dart throwing algorithm [Cook 1986], obtain initial particle velocities from the velocity field, and run position-based fluid simulation [Macklin and Müller 2013] to simulate those particles slightly forward in time. We treat those simulated particles as missing droplets caused by early ruptures at splash fronts and we voxelize them to the two fields, so that they can naturally merge with the bulky volume. Fig. 10 compares reconstructed surface mesh results before and after this particle-based refinement process, and Fig. 11 demonstrates how this process improves the visual quality of a milk skirt example. We note that the use of particle-based simulation is to keep those particles together under surface tension as a single droplet. Without it, advected particles disperse after they leave the bulky volume.



(a) Before refinement (b) After refinement



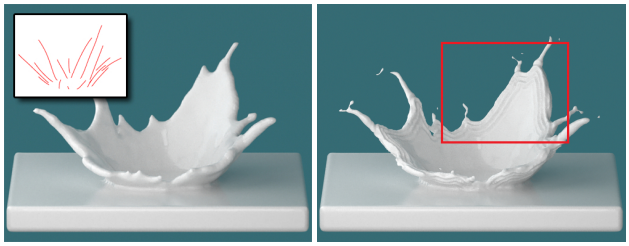
(c) Before refinement (d) After refinement

Fig. 11. A milk skirt example. In this example, the user draws strokes to create two milk skirts in different configurations dressing the same dancing sculpture model. We apply particle-based refinement only in this example.

The described process so far is for adding extra droplets to the synthesized model. To make the splash front more rupture-like, we also provide an inverse process for removing front regions. Basically, we pick random sample points in air: $\phi_{\text{air}} - \phi_{\text{band}} \leq \phi(\mathbf{q}_i) < \phi_{\text{air}}$, generate air particles around them, and then simulate them backward with interpolated velocities from the field. After that, we voxelize those particles and carve the scalar field ϕ accordingly.

5.2 Mesh-Based Refinement

Our system offers an optional refinement process, after the liquid mesh gets reconstructed from volumetric output. The purpose of this process is to generate capillary waves, which are triggered at splash fronts and propagated by surface tension as shown in Fig. 12d. The reason we perform this refinement on meshes, rather than on particles [Yang et al. 2016], is because wave seeding and propagation is more straightforward to implement on meshes.



(a) Before refinement (b) After refinement

(c) A closeup view (d) A photograph

Fig. 12. The surface mesh results before and after refinement. The mesh-based refinement process generates capillary wave effects near splash fronts, as shown in (b) and (c). The system allows the user to control these waves by various simulation variables. The inset image in (a) shows the user strokes.

5.2.1 Wave seeding. Our first question is: where do capillary waves start? When enriching a simulation sequence, Yang et al. [2016] initialized capillary waves as surface tension energy changes. Unfortunately, we cannot do the same thing here, since we are dealing with a static surface mesh. Instead we generate capillary waves near high curvature regions, and similar to [Liu et al. 2006a], we assume that they are sinusoidal. Let ρ_i^t be the wave intensity of vertex i at a pseudo time instant t. We calculate $\hat{\rho}_i^{t+\Delta t}$, the initialization of $\rho_i^{t+\Delta t}$ at time $t + \Delta t$ by:

$$\hat{\rho}_i^{t+\Delta t} = \rho_i^t + \kappa_i \left(\sin(\omega t + \omega \Delta t) - \sin(\omega t)\right), \tag{5}$$

in which Δt is the pseudo time step, ω is the wave frequency constant, κ_i is the mean curvature and $\hat{\rho}_i^0 = 0$. We note that κ_i can be negative at vertex i near concave fronts.

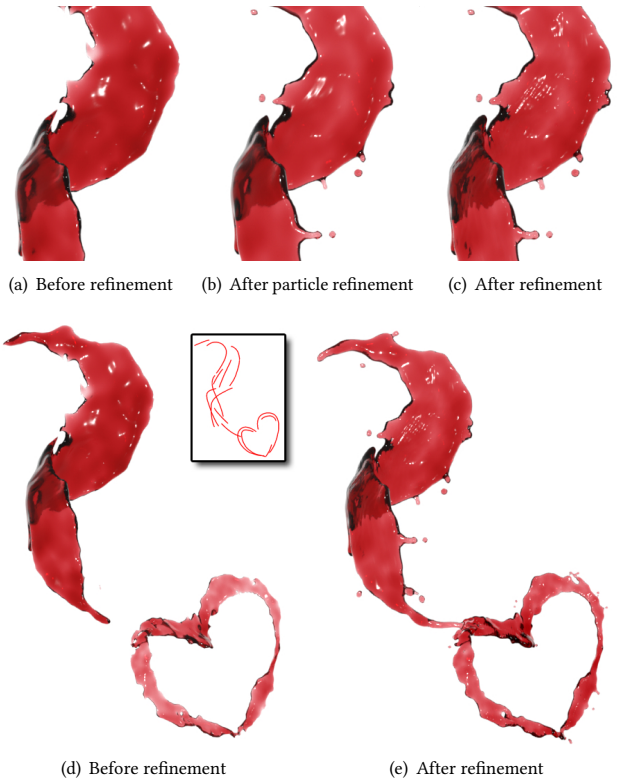
5.2.2 Wave propagation. Similar to [Yang et al. 2016], we model capillary wave propagation by the acoustic wave equation:

$$\frac{d^2\rho}{dt^2} = c_{\text{wave}}^2 \nabla^2 \rho_i, \tag{6}$$

in which c_{wave} is the wave speed. To solve this equation, we introduce another variable $\dot{\rho}_i$ representing the wave intensity changing rate at vertex i. We then use Leapfrog integration to update ρ_i and $\dot{\rho}_i$:

$$\begin{cases} \rho_i^{t+\Delta t} = \hat{\rho}_i^{t+\Delta t} + \Delta t \dot{\rho}_i^t, \\ \dot{\rho}_i^{t+\Delta t} = \zeta \dot{\rho}_i^t + \Delta t c_{\text{wave}}^2 \nabla^2 \rho_i^{t+\Delta t}, \end{cases} \tag{7}$$

where ζ is the damping coefficient. We use the discrete Laplacian operator to discretize and solve Eq. 7. To ensure numerical stability, we need a sufficiently small pseudo time step Δt that satisfies the



(a) Before refinement (b) After particle refinement (c) After refinement

(d) Before refinement (e) After refinement

Fig. 13. A wine heart example. In this example, the user creates a heart-shaped liquid splash as if it was wine pouring out of a bottle. This model is made of two splashes: the streamlet and the heart. Particle-based and mesh-based refinement processes improve the visual quality of this example.

underlying CFL condition. In our experiment, we simply reduce Δt until the simulation becomes stable.

5.2.3 Wave simulation and mesh update. Given the presented wave seeding and propagation methods, we run wave simulation over the reconstructed mesh for a short period of time and use the wave intensity to adjust the position of every vertex in its normal direction. In practice, we often have to subdivide the reconstructed mesh ahead of time, so that the resolution can be high enough for modeling discretized waves. The pseudo simulation time controls how far capillary waves travel: the longer the simulation is, the farther the waves reach. Fig. 12 and 13 compare the mesh results of two examples before and after mesh-based refinement.

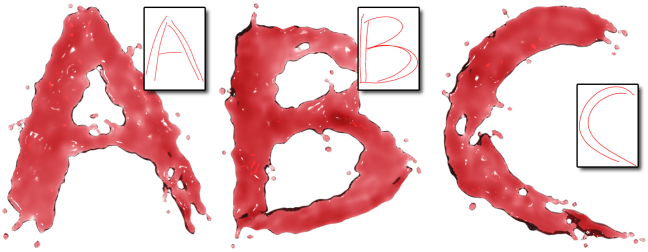
We note that capillary waves are not so noticeable at a large scale. Even at a small scale, the number, the magnitude and the frequency of capillary waves vary dramatically from liquid to liquid, according to liquid physical properties. Our system asks the user to decide the use of the mesh-based refinement process as part of model design choices, through the adjustment of wave simulation variables.

5.3 Quantitative Analysis

Table 1 also quantitatively evaluates our refinement result in comparison with the ground truth by the four metrics, using the same refinement parameter values in Fig. 13. It shows that the use of our refinement processes increases both TPR and FPR, while decreases IoU. This outcome is expected, because our refinement processes tend to add volume to synthesized splash shapes. But overall, the influence of refinement processes on the result by these metrics is small, thanks to those small-scale shape changes only.

Table 2. The parameters and their values used by model refinement.

Label	Meaning	Value
ϕ_{air}	Air-liquid interface threshold	0.5
ϕ_{band}	Air-liquid interface bandwith	0.4
α	The first exponential variable	2 to 8
β	The second exponential variable	2 to 8
γ	Sample selection threshold	By user
ω	Capillary wave frequency	1 to 10Hz
c_{wave}	Capillary wave speed	0.1 to 1.0m/s
ζ	Capillary wave damping coefficient	0.9



(a) Splash "A" (b) Splash "B" (c) Splash "C"

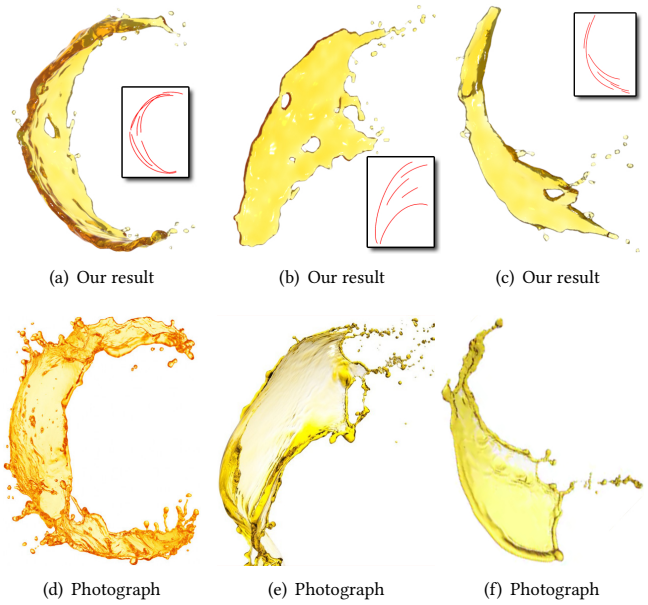
Fig. 14. Wine splashes in alphabetic shapes. The user creates those quality shapes by drawing only two to five strokes.

6 RESULTS

We implement our interactive system and test it on an Intel Core i7-5930K 3.5GHz CPU and an NVIDIA GeForce GTX TITAN X GPU. Our system uses an Oculus Rift S VR gaming headset for displaying the virtual reality environment and its handheld motion controller for sketch input. The total computational time of the splash model synthesis component, including both the network inference time and the mesh reconstruction time, is under 0.2s. Therefore, the user can examine the result and edit his/her strokes interactively on the fly, until he/she is satisfied with the result. After that, the user can apply the offline splash model refinement component provided by our system to improve splash shape details. The total computational time of the refinement component is between 0.2s and 0.4s, depending on the particle number and the mesh resolution. Table 1 summarizes the parameters and their values used by model refinement.

6.1 Special Effects

The user can also apply our system to create desired special splash effects. Fig. 1 shows two liquid butterflies composed of multiple liquid splashes, and Fig. 13 shows a heart-shaped splash as if it



(a) Our result (b) Our result (c) Our result

(d) Photograph (e) Photograph (f) Photograph

Fig. 15. Our results in comparison with photographs. Our system allows the user to create liquid splashes in different shapes, as shown from (a) to (c). The user controls splash shapes by stroke trajectories and stroke velocities.

was formed by pouring wine. In the past, a photographer needs to composite multiple photographs to create image results with similar special effects. In our system, the user can interactively draw many sets of strokes for multiple splashes and check the overall model result on the fly. Fig. 11 demonstrates a milk skirt example, which provides two milk skirts dressing a dancing human sculpture. The user draws different strokes to control skirt shapes. Finally, Fig. 14 and 17 show liquid splashes in alphabetic shapes, intentionally designed by our user.

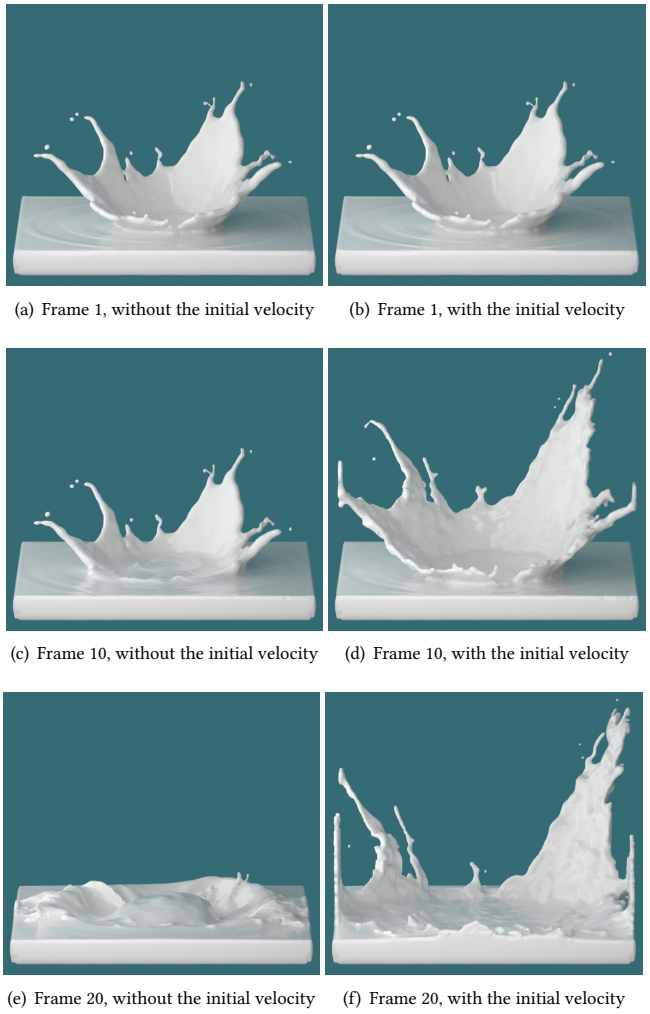
6.2 Comparison with Photographs

To test the capability of our system in generating specific liquid splash shapes, we ask the user to draw strokes that match with the shapes in photographs from Fig. 15d to 15f and we run our system to obtain the results shown from Fig. 15a to 15c. As discussed in Section 5, it is difficult for our system to provide as many rich details as in photographs. But in general, our results are consistent with user expectations and they can serve as pre-visualization for additional editing and refinement. We note that setting up the initial environment by hand for specific shapes is difficult in both the real world and the virtual world.

6.3 Physics-Based Fluid Animation

Compared with splash photographs in 2D and manually modeled shapes in 3D, our splash model has a unique advantage: it contains both the shape and the velocity. Therefore we can treat the model as the initial condition and run physics-based simulation immediately, without resorting to additional tools for injecting the initial velocity. In Fig. 16, we show the animation result of a milk crown after a streamlet hits a bulky liquid volume, by treating our model in Fig. 16b as the initial frame. In Fig. 17, we show the animation result of a specifically designed liquid splash in the "S" shape, by treating our model in Fig. 17b as the initial frame. Without the synthesized velocity field, the results would be inconsistent with the expected splash events, as shown in Fig. 16 and 17.

We note that the synthesized velocity field may not be divergence-free. Fortunately, this is not a practical issue in animation production, since the divergence of the velocity field is quickly eliminated by pressure projection in the next time step. For applications that do require divergence-free velocity field inputs, we can post-process the synthesized velocity field by pressure projection as well.



(a) Frame 1, without the initial velocity (b) Frame 1, with the initial velocity

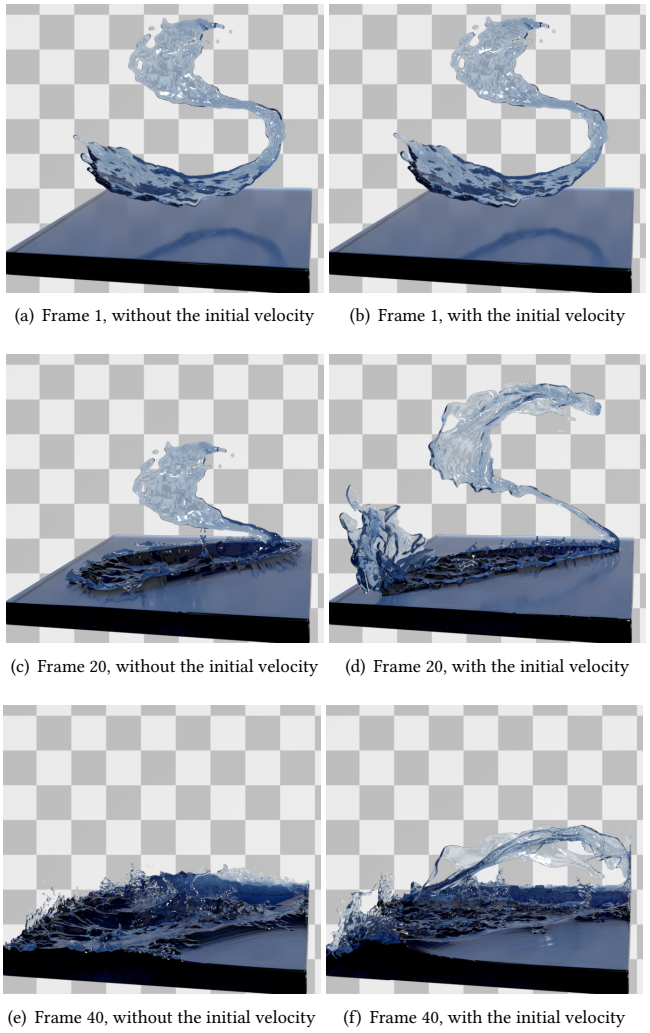
(c) Frame 10, without the initial velocity (d) Frame 10, with the initial velocity

(e) Frame 20, without the initial velocity (f) Frame 20, with the initial velocity

Fig. 16. An animated milk crown example without and with the initial velocity. Without the initial velocity, the milk crown quickly collapses under gravity as shown in (a), (c) and (e). With the initial velocity, the splash moves forward in the splash front direction as shown in (b), (d) and (f).

6.4 User Studies

We conduct two user studies to evaluate the visual quality and the usability of our system.



(a) Frame 1, without the initial velocity (b) Frame 1, with the initial velocity

(c) Frame 20, without the initial velocity (d) Frame 20, with the initial velocity

(e) Frame 40, without the initial velocity (f) Frame 40, with the initial velocity

Fig. 17. An animated "S" example without and with the initial velocity. Without the initial velocity, the liquid splash in the "S" shape falls straight into the surface as shown in (a), (c) and (e). With the initial velocity, the "S" splash expands as shown in (b), (d) and (f).

6.4.1 Visual quality evaluation. To evaluate the visual result quality of our system, we adopt Google Forms to recruit seven male and seven female users, whose ages range from 21 to 63 and whose education levels are from high school diploma to Doctoral degree. In this study, we present two splash images to each user every time, one generated by physics-based simulation and one generated by our system, and ask each user to decide which one looks more visually plausible. In total, we give each user 16 image pairs to choose from. The study shows that the simulation result is chosen over our result 55.8 percent of the time, while our result is chosen 44.2 percent of the time. This experiment shows that even though our result is different from the simulation result, mostly because of fewer small-scale details, we can still consider it to be visually plausible overall.

6.4.2 Usability evaluation. Next we evaluate the usability of our system in an immersive virtual reality environment and we compare it with other modeling tools, i.e., the X-Particles system by Cinema 4D and the mesh sculpting system by Maya and ZBrush. We recruit eight users, who have five-year 3D modeling experience on average. In this study, we ask the users to design a liquid splash in the "S" shape. Using our system, they can interactively and conveniently model their splashes within one minute, as shown in Fig. 18a. The simulation-based particle system is also easy to use, but due to a lack of intuitive connection between initial conditions and simulation outcomes, it is difficult to obtain satisfactory results as shown in Fig. 18b. On average, the users spend 20 minutes with this tool. Finally, the mesh sculpting system is a powerful tool with which the users can create any shape they want, given sufficient time. Unfortunately, the modeling process by this tool is tedious, as the users have to address every surface detail by hand for realistic mesh results. We ask the users to spend at least one hour with this tool and the results are just comparable to ours, as shown in Fig. 18c.

6.5 Limitations

Perhaps the greatest limitation of our system is the difficulty of generating rich details comparable to those in real-world splashes. The model refinement processes help lessen this issue, but they do not solve it. While our system enables the user to design splash models by simple strokes, it does not offer any additional control other than post-processing. This issue can be problematic, given the fact that cGANs, unlike GANs, are in short of stochasticity [Isola et al. 2017] and the user cannot expect largely different model results to choose from. If the results are not aesthetically appealing or consistent with their design goals, the user has to treat the system as a black box and modify the strokes through trials and errors. Currently, our system is more suitable for designing freeform splash models. When there are constraints or obstacles in the environment, such as the dancing human sculpture in Fig. 11, the system relies on the user to adjust the strokes accordingly and it cannot enforce constraints in an automatic way. Our current system is also unsuitable for designing liquid models other than splashes and the result can be unpredictable if the strokes are not streamline-like, such as those self-intersecting ones in Fig. 19. Finally, our system takes 3D user strokes acquired by depth-augmented devices only. It is unclear how the system responds when it takes other strokes as input, but we can always run a pre-processing step to convert other strokes into 3D ones, as a quick way of generalization.

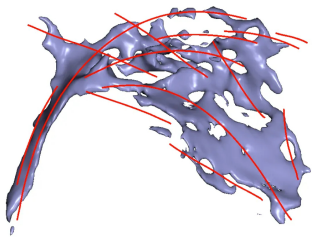
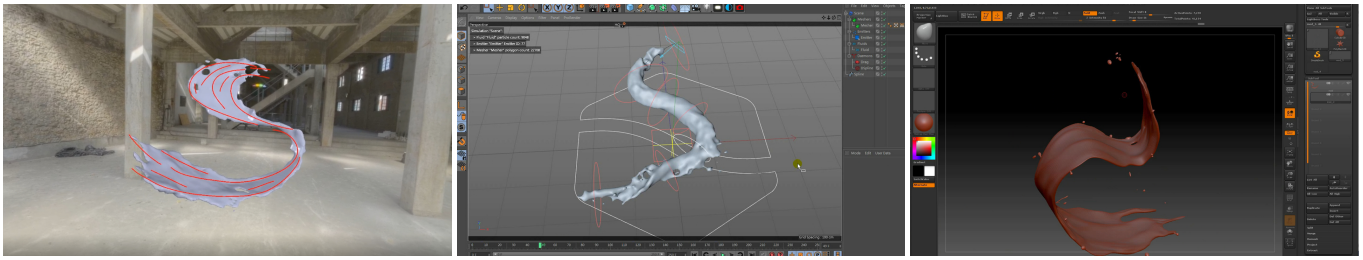


Fig. 19. A failure case. In this example, our synthesizer fails to produce a plausible splash results due to self-intersecting strokes.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we demonstrate the effective use of a conditional generative adversarial network (cGAN) in synthesizing complex



(a) Our system in a virtual reality environment (b) The X-Particles system by Cinema 4D (c) The mesh sculpting systems by Maya and ZBrush

Fig. 18. The interfaces of three modeling systems. Our system allows users to interactively and conveniently create quality liquid shape models in a very short time. In comparison, other modeling systems require users to spend a sufficient amount of time before obtaining acceptable splash results.

liquid splash models from user strokes. To overcome the missing detail issue in comparison with real-world splashes, we develop a physics-inspired splash model refinement component for the user to enrich synthesized results. Thanks to the efficiency of our system, the user can now interactively edit and examine his/her results, in the course of designing special splash effects.

Looking in the future, we set our priority as further detail refinement. This includes collecting more diversified and detailed data, improving neural network structures, and better model refinement processes. We then would like to improve the runtime performance of our system, especially by its GPU implementation, to eventually achieve real-time model design and synthesis. How to optimize user experience and achieve precise shape design in a virtual environment is another important problem we will study. Finally, we will explore the extension of our system for modeling other liquid phenomena, some of which are also compatible with the streamline representation and can be synthesized in a similar fashion.

ACKNOWLEDGMENTS

The authors would like to thank Nvidia and Adobe for their funding and equipment support.

REFERENCES

Jeremiah Uhler Brackbill, Douglas B. Kothe, and Hans Max Ruppel. 1988. Flip: A Low-Dissipation, Particle-in-Cell Method for Fluid Flow. In *the Workshop on Particle Methods in Fluid Dynamics and Plasma Physics.*

José A. Canabal, David Miraut, Nils Thuerey, Theodore Kim, Javier Portilla, and Miguel A. Otaduy. 2016. Dispersion Kernels for Water Wave Simulation. *ACM Trans. Graph.* 35, 6, Article 202 (Nov. 2016), 10 pages.

Mengyu Chu and Nils Thürey. 2017. Data-Driven Synthesis of Smoke Flows with CNN-Based Feature Descriptors. *ACM Trans. Graph. (SIGGRAPH)* 36, 4 (2017), 1–14.

Jonathan M Cohen, Lee Markosian, Robert C Zeleznik, John F Hughes, and Ronen Barzel. 1999. An Interface for Sketching 3D Curves. In *Proceedings of I3D.* 17–21.

Robert L. Cook. 1986. Stochastic Sampling in Computer Graphics. *ACM Trans. Graph.* 5, 1 (Jan. 1986), 51–72.

Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2016. Surface-Only Liquids. *ACM Trans. Graph. (SIGGRAPH)* 35, 4, Article 78 (July 2016), 12 pages.

Chris De Paoli and Karan Singh. 2015. SecondSkin: Sketch-Based Construction of Layered 3D Models. *ACM Trans. Graph. (SIGGRAPH)* 34, 4 (2015), 1–10.

Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A. Efros, and Adrien Bousseau. 2018. 3D Sketching Using Multi-View Deep Volumetric Prediction. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 21 (July 2018), 22 pages.

Nick Foster and Dimitri Metaxas. 1996. Realistic Animation of Liquids. *Graphical models and image processing* 58, 5 (1996), 471–483.

Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018. GPU Optimization of Material Point Methods. *ACM Trans. Graph. (SIGGRAPH)* 37, 6, Article 254 (Dec. 2018), 12 pages.

Tony Generico. 2017. *Splash: High-Speed Photography With Liquids.* CreateSpace Independent Publishing.

Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured Annotations for 2D-to-3D Modeling. In *ACM SIGGRAPH Asia 2009 Papers.* Article 148, 9 pages.

Éric Guérin, Julie Digne, Éric Galin, Adrien Peytavie, Christian Wolf, Bedrich Benes, and Benoît Martinez. 2017. Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks. *ACM Trans. Graph.* 36, 6, Article 228 (Nov. 2017), 13 pages.

Zhongyuan Hu, Haoran Xie, Tsukasa Fukusato, Takahiro Sato, and Takeo Igarashi. 2019. Sketch2VF: Sketch-Based Flow Design with Conditional Generative Adversarial Network. *Computer Animation and Virtual Worlds* 30, 3-4 (2019), 1889.

Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. 2016. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE transactions on visualization and computer graphics* 23, 8 (2016), 2003–2013.

Ruoguan Huang, Zeki Melek, and John Keyser. 2011. Preview-Based Sampling for Controlling Gaseous Simulations. In *Proceedings of SCA.* 177–186.

Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques.* 409–416.

Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. 2012. Unified Spray, Foam and Air Bubbles for Particle-Based Fluids. *The Visual Computer* 28 (2012), 669–677.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* (2015). arXiv:1502.03167

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 1125–1134.

Stefan Jeschke and Chris Wojtan. 2017. Water Wave Packets. *ACM Trans. Graph. (SIGGRAPH)* 36, 4, Article 103 (July 2017), 12 pages.

Byungsoo Kim, Vinicius C Azevedo, Nils Thürey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum (Eurographics)* 38, 2 (2019), 59–70.

Theodore Kim, Jerry Tessendorf, and Nils Thürey. 2013. Closest Point Turbulence for Liquid Surfaces. *ACM Trans. Graph.* 32, 2, Article 15 (April 2013), 13 pages.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR.*

LâĂŹubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-Driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.* 34, 6, Article 199 (Oct. 2015), 9 pages.

Jeehyung Lee and Thomas A Funkhouser. 2008. Sketch-Based Search and Composition of 3D models. In *Proceedings of SBM.* 97–104.

Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2017. BendSketch: Modeling Freeform Surfaces through 2D Sketching. *ACM Trans. Graph. (SIGGRAPH)* 36, 4, Article 125 (July 2017), 14 pages.

Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2018. Robust Flow-Guided Neural Prediction for Sketch-Based Freeform Surface Modeling. *ACM Trans. Graph.* 37, 6, Article 238 (Dec. 2018), 12 pages.

Tee Tai Lim and Alexander J. Smits. 2000. *Flow Visualization: Techniques and examples.* Imperial College Press.

Shengjun Liu, Xiaogang Jin, Charlie CL Wang, and Jim X Chen. 2006a. Water-Wave Animation on Mesh Surfaces. *Computing in Science & Engineering* 8, 5 (2006), 81–87.

Zhanping Liu, Robert Moorhead, and Joe Groner. 2006b. An Advanced Evenly-Spaced Streamline Placement Algorithm. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 965–972.

Miles Macklin and Matthias Müller. 2013. Position Based Fluids. *ACM Trans. Graph. (SIGGRAPH)* 32, 4, Article 104 (July 2013), 12 pages.

Pierre-Luc Manteaux, Ulysse Vimont, Chris Wojtan, Damien Rohmer, and Marie-Paule Cani. 2016. Space-Time Sculpting of Liquid Animation. In *Proceedings of the 9th International Conference on Motion in Games*. 61–71.

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. In *ACM SIGGRAPH 2004 Papers*. 449–456.

Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. 2015. Surface Turbulence for Particle-Based Liquid Simulations. *ACM Trans. Graph.* 34, 6, Article 202 (Oct. 2015), 10 pages.

Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014). arXiv:1411.1784

Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: Designing Freeform Surfaces with 3D Curves. In *ACM SIGGRAPH 2007 papers*. 41–50.

Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. 2005. A Sketch-Based Interface for Detail-Preserving Mesh Editing. In *ACM SIGGRAPH 2005 Papers*. 1142–1147.

Michael B. Nielsen and Robert Bridson. 2011. Guide Shapes for High Resolution Naturalistic Liquid Simulation. In *ACM SIGGRAPH 2011 Papers*. Article 83, 8 pages.

Stanley Osher and Ronald Fedkiw. 2003. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag.

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive Localized Liquid Motion Editing. *ACM Trans. Graph.* 32, 6, Article 184 (Nov. 2013), 10 pages.

Zherong Pan and Dinesh Manocha. 2017. Efficient Solver for Spacetime Control of Smoke. *ACM Trans. Graph. (SIGGRAPH)* 36, 5, Article 162 (July 2017), 13 pages.

Nick Rasmussen, Douglas Enright, Duc Nguyen, Sebastian Marino, Nigel Sumner, Willi Geiger, Samir Hoon, and Ronald Fedkiw. 2004. Directable Photorealistic Liquids. In *Proceedings of SCA*. 193–202.

Karthik Raveendran, Nils Thürey, Chris Wojtan, and Greg Turk. 2012. Controlling Liquids Using Meshes. In *Proceedings of SCA*. 255–264.

Karthik Raveendran, Chris Wojtan, Nils Thürey, and Greg Turk. 2014. Blending Liquids. *ACM Trans. Graph. (SIGGRAPH)* 33, 4, Article 137 (July 2014), 10 pages.

Alec Rivers, Frédo Durand, and Takeo Igarashi. 2010. 3D Modeling with Silhouettes. *ACM Trans. Graph. (SIGGRAPH)* 29, 4, Article 109 (July 2010), 8 pages.

Lior Rokach and Oded Maimon. 2005. Clustering Methods. In *Data mining and knowledge discovery handbook*. Springer, 321–352.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Vol. 9351. 234–241.

Bruno Roy, Eric Paquette, and Pierre Poulin. 2020. Particle Upsampling as a Flexible Post-Processing Approach to Increase Details in Animations of Splashing Liquids. *Computer & Graphics* 88 (2020), 57–69.

Ryan Schmidt and Karan Singh. 2008. Sketch-Based Procedural Surface Modeling and Compositing Using Surface Trees. In *Computer Graphics Forum*, Vol. 27. 321–330.

Lin Shi and Yizhou Yu. 2005. Controllable Smoke Animation with Guiding Objects. *ACM Trans. Graph.* 24, 1 (Jan. 2005), 140–164.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958.

Nils Thürey, Richard Keiser, Mark Pauly, and Ulrich Rüde. 2009. Detail-Preserving Fluid Control. *Graphical Models* 71, 6 (2009), 221–228.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating Eulerian Fluid Simulation with Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. 3424–3433.

Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. 2003. Keyframe Control of Smoke Simulations. *ACM Trans. Graph. (SIGGRAPH)* 22, 3 (July 2003), 716–723.

Kiwon Um, Xiangyu Hu, and Nils Thürey. 2018. Liquid Splash Modeling with Neural Networks. *Computer Graphics Forum* 37, 8 (2018), 171–182.

Nobuyuki Umetani and Bernd Bickel. 2018. Learning Three-Dimensional Flow for Interactive Aerodynamic Design. *ACM Trans. Graph. (SIGGRAPH)* 37, 4, Article 89 (July 2018), 10 pages.

Steffen Wiewel, Moritz Becher, and Nils Thürey. 2019. Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. In *Computer Graphics Forum*, Vol. 38. 71–82.

Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. 2010. Physics-Inspired Topology Changes for Thin Fluid Features. In *ACM SIGGRAPH 2010 Papers*. Article 50, 8 pages.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. TempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow. *ACM Trans. Graph. (SIGGRAPH)* 37, 4, Article 95 (July 2018), 15 pages.

Kun Xu, Kang Chen, Hongbo Fu, Wei-Lun Sun, and Shi-Min Hu. 2013. Sketch2Scene: Sketch-Based Co-Retrieval and Co-Placement of 3D Models. *ACM Trans. Graph. (SIGGRAPH)* 32, 4, Article 123 (July 2013), 15 pages.

Sheng Yang, Xiaowei He, Huamin Wang, Sheng Li, Guoping Wang, Enhua Wu, and Kun Zhou. 2016. Enriching SPH Simulation by Approximate Capillary Waves. In *Proceedings of SCA*. 29–36.

Bo Zhu, Michiaki Iwata, Ryo Haraguchi, Takashi Ashihara, Nobuyuki Umetani, Takeo Igarashi, and Kazuo Nakazawa. 2011. Sketch-Based Dynamic Illustration of Fluid Systems. *ACM Trans. Graph. (SIGGRAPH Asia)* 30, 6 (Dec. 2011), 1âĂŞ8.

Yongning Zhu and Robert Bridson. 2005. Animating Sand as a Fluid. *ACM Trans. Graph. (SIGGRAPH)* 24, 3 (July 2005), 965–972.