Particle-Based Fluid Surface Rendering with Neural Networks

Viktória Burkus
Budapest University of Technology and Economics
Budapest, Hungary
burkus@iit.bme.hu

Attila Kárpáti
Budapest University of Technology and Economics
Budapest, Hungary
karpati@iit.bme.hu

László Szécsi
Budapest University of Technology and Economics
Budapest, Hungary
szecsi@iit.bme.hu

ABSTRACT

Surface reconstruction for particle-based fluid simulation is a computational challenge on par with the simulation itself. In real-time applications, splatting-style rendering approaches based on forward rendering of particle impostors are prevalent, but they suffer from noticeable artifacts.
In this paper, we present a technique that combines forward rendering simulated features with deep-learning image manipulation to improve the rendering quality of splatting-style approaches to be perceptually similar to ray tracing solutions, circumventing the cost, complexity, and limitations of exact fluid surface rendering by replacing it with the flat cost of a neural network pass. Our solution is based on the idea of training generative deep neural networks with image pairs consisting of cheap particle impostor renders and ground truth high quality ray-traced images.

Keywords

Computer Graphics, Metaballs, Generative Neural Network, Surface Reconstruction

1 INTRODUCTION

Realistic and resource-efficient visualization of fluid surfaces reconstructed with particle-based simulations is an ongoing challenge in modern computer graphics. The thousands of particles required to create plausible fluid behavior significantly complicate the execution of algorithms such as recursive ray tracing, which, due to their complexity, can often only be used to display simulation results in real-time using accelerating methods.

Representing particles with metaballs is a widely used and popular method for displaying the results of particle-based simulations. The metaball construct, introduced by Blinn [Bli82] and Nishimura [Nis85], describes an implicit surface formed by interacting objects. Each metaball has a radial density function (or radial basis function, RBF) and the metaballs together represent the isosurface of the density field. Due to the temporal change in the simulation, it is necessary to evaluate the isosurface in each frame, which significantly limits its usability in real-time applications. Since their inception, several publications have been produced to speed up the visualization of metaballs.

Grid-based solutions usually give faster results when the grid resolution is low, but artifacts form on the liquid surface. The disadvantage of solutions based on *on-surface tracer particles* is the complexity of maintaining a uniform coverage and the difficulty of avoiding gaps that form on the fluid surface. The problem to be solved in *ray tracing-based methods* is to speed up the evaluation of the density function required for the computation of the intersection between a ray and the surface. All of these methods are useful when a larger number of metaballs needs to be visualized, but realistic shading or global illumination requires additional resources in a different order of magnitude.

Neural networks trained with input exemplar–expected output image pairs have achieved significant success in the field of image manipulation. This paper contributes to the field of neural rendering by proposing a hybrid rendering solution for particle-based fluid simulation, combining stochastic rendering and image-to-image neural networks. This can be used in any context where fast fluid surface rendering is needed, but certain lighting effects require the environment to be fixed at training time.

Metaballs with complex shading are achieved by introducing image enhancement using a neural network, trained with the method presented in our paper. The question we investigate is how to generate the input images and the expected target images needed to train the

network to obtain a satisfactory result. We also analyze emerging artifacts to motivate further research.

The paper is organized as follows. Section 2 presents previous work on metaball rendering (not using neural networks), and the state of the art on neural rendering in general. Section 3 describes the proposed method. The results are summarized in Section 4, and finally, the conclusion and further research directions are described in Section 6 and Section 5.

2 PREVIOUS WORK

The first particle-based implicit surface was presented by J.F. Blinn [Bli82], which involved displaying electron density maps of molecular structures.

Blinn used a linear combination of Gaussian *radial basis functions* (RBFs) and a constant term for constructing the implicit equation of the surface. As Gaussian RBFs do not have finite support, all objects must be considered in order to determine the function value at any point, which is a necessary operation in any method extracting an isosurface. Therefore, its application involves considerable computational effort. Replacing the Gaussian density function with a finite-support function can reduce the computational effort considerably, since to evaluate the total density function at a point it is sufficient to consider the contribution of the metaballs within the finite environment of this spatial point. For our experiments in this paper, we used the sixth-degree polynomial RBF proposed by Wyvill [WMW86].

2.1 Metaball rendering techniques

The previous work on the representation of the surface of metaballs can be divided into several groups. However, the aim of all the publications mentioned below is to accelerate the evaluation of the density function. In this section, we survey these methods to highlight the fact that existing methods are either expensive or specialized with respect to the light transport phenomena they support.

2.1.1 Grid-based solutions

A widely used method is the *marching cubes* algorithm presented by Lorensen et al. [LC87]. The algorithm generates a triangular mesh model approximating an isosurface of volumetric density function. Such a model can be ray-traced efficiently using conventional methods. However, the quality of the generated surface depends on the resolution of the grid used. If a low-resolution grid is used, the algorithm is able to generate the surface quickly, but the resolution of the surface is low. When a high-resolution grid is used, the algorithm generates a detailed surface, but at increased computational and memory costs. In particular, the density function has to be evaluated at a high number of grid nodes.

Krone et al. [KSES12] present a GPU-accelerated version of the marching cubes approach. Even though they work with Gaussian density kernels, they use a cutoff radius to decrease complexity. Sorting into a 3D auxiliary grid is used for proximity searches.

The approach of Iwasaki et al. [IDYN06] makes use of a grid, but it is constructed using a virtual camera, and rendering metaballs, clustered into layers, into the grid. The method is strongly oriented towards configurations where most of the fluid forms a roughly horizontal surface in a container, with very few layers of splashes above. The isofurface is rendered using a splatting-like method named *surfels*. Real-time performance is achieved.

All grid based approaches are ill-suited for scenarios where the fluid cannot be assumed to reside within a finite container.

2.1.2 Screen-space visualization

Wladimir J. van der Laan et al. [vdLGS09] presented a screen-space metaball visualization method, which relies on rendering the metaballs as solid spheres, and applying image-space filtering to smooth the surface. The method provides real-time performance with a configurable speed-quality preference, but it is only efficient if the spheres provide a close approximation of the surface indeed, i.e. if only a few particles affect a surface point. Larger reconstruction kernels encompassing higher number of particles cannot be reproduced, and thus the fluid retains a somewhat viscous appearance even with the best filtering, especially when the balls appear large in image space.

Müller et al. [MGE07] also used metaballs for the visualization of a molecular dynamics simulation. They propose two ideas. The *vicinity texture* is a precomputed lookup structure for metaball neighbors, which is suitable for static geometries, but requires high texture bandwidth still. The *walking depth plane* approach finds isosurface depth in pixels using iterative correction of an initial guess. This, however, requires the metaballs to be rendered as billboards up to 100 times, making the approach scale poorly both with respect to the number of particles and to the target resolution.

Xu et al. [XW16] makes use of hardware multisampling anti-aliasing and alpha-to-coverage to perform four-layered depth peeling in a single pass. Using multiple such passes, a polynomial approximation of the density function using the first few metaballs can be obtained and solved explicitly. The approach works fast and provides accurate results when only a few metaballs contribute to individual surface points. Despite the ingenuity and performance delivered, the acceleration still only works for primary rays.

Van Kooten et al. [KBT07] reconstruct the surface using on-surface tracer particles uniformly distributed on the fluid surface. Repulsive forces, velocity constraints, and particle densities are used to determine the position of the particles, which are travelling on the surface following its motion. The expensive part of the method is computing the forces, especially maintaining the data structure to find near tracer particles. Where maintaining uniform coverage fails, small gaps may appear on the surface.

Kanamori et al. [KSN08] present a method based on ray tracing. Using depth peeling to obtain boundaries of effective spheres, polynomials for the density along ray segments are obtained and solved using the Bezier clipping root finding method. The method can produce high quality images, but the root finding is somewhat expensive.

Szécsi et al. [SI12] present a similar scheme based on building per-pixel fragment lists. They render particles as billboards, storing an entry, midpoint, and exit fragment in a per-pixel list. By sorting these fragments, and finding a cubic density approximation on ray segments between them, they are able to perform ray-surface intersection in a less expensive fashion. The drawback is the high overhead of gathering the per-pixel list.

2.1.3 Ray-casting and ray-marching methods

All screen-space methods fail to obtain correct reflections and refractions efficiently, as they organize their data structure around processing primary rays. An environment mapping approximation, neglecting self-reflections, multiple refractions, and media participation, is acceptable in some, but not all configurations.

Accurate visualization requires recursive ray tracing, where the evaluation of the density function at arbitrary points must be very efficient. Recently, Winchenbach and Kolb [WK20] proposed a GPU-friendly data structure that offers exceptional performance. However, the approach is still far from real-time.

2.2 Deep learning

The problem we investigate in this paper is a neural rendering problem, in the sense that an image should be produced from an RBF volume representation. As we use conventional rendering combined with a neural network, ours is not a pure neural rendering approach.

Pure text-to-image efforts are reviewed Frolov et al. [FHR+21], and today they are capable of synthesizing images sampled from a space spanned by training photo databases, but not from abstract geometrical scene descriptions.

Tewari et al. [TFT+20] summarize and classify the state-of-the-art neural rendering approaches that combine classical computer graphics pipelines and neural networks. Notably absent for the results are solutions that would render deterministic specular effects like reflections or refractions.

Thomas and Forbes [TF17] managed to achieve fast global illumination with deep neural networks, featuring diffuse interreflections with detailed occlusion, but without reflective or refractive materials.

Feygina et al. [FIM18] introduced CycleWGAN networks for enhancing global illumination rendering in post-processing for computer games, showing that the deep learning approach can be viable in real time.

Constantin and Bigand [CCB20] handle scenes with reflections and refractions, but they only use the neural network to detect noise and thus direct sampling in a path tracing algorithm.

Bui et al. [BLMD18] present a convolutional neural network-based approach for rendering high-resolution images from a point cloud.

Horvath [Hor19] uses a conditional GAN to generate metaball images from a simplified set of input training data. For the input, four-channel (rgb, depth) images of circles are rendered, where every circle represents a metaball. For output, it generates images of metaballs consisting of a color buffer, a depth buffer, and a normal buffer. No shadows, reflections, or refractions are produced. Our approach in this paper is going to be similar, but our objective is rendering images with perceptively acceptable reflections and reflections.

As we assume a volume representation based on RBFs, it is of interest to mention RBFNNs, or Radial Basis Function Neural Networks [BL88], and the more recent deep-RBF neural networks [ZHS18]. These contain a layer that outputs positions and weights for RBFs that reconstruct the function whose samples were used as training data. It has been used successfully in the function approximation and classification domains, but they have not been applied to rendering so far. In our work, the RBF representation of the fluid is considered an input, not an output—it is created by conventional physical simulation, not by a neural network. The output, in our case, is a rendered image. Therefore, our solution may be used in conjunction with deep-RBF networks producing 3D density representations.

Overall, high quality metaball rendering methods capable of rendering reflections, refractions, and media participation, are not real-time for a reasonable number of particles. Fast methods rely on some kind of screen space accumulation, and offer poor substitutes for reflection and refraction rendering. Note that our work is based on none of the above techniques. Instead, we aim to stylize an inexpensive render of particles into one featuring reflections and refractions using trained image-to-image transfer with a deep neural network. In part, this is similar to the van der Laan

approach [vdLGS09] of image space filtering, but using learned kernels instead of constructed ones. As above methods using depth listing or depth peeling, we also strive to gather information about multiple layers of metaballs, but we exploit the idea of stochastic rendering. In this, our solution is also different from Horvath's work [Hor19], and we also extend the approach to rendering reflections and refractions.

Our solution is based on Pix2Pix [IZZE17] and CycleGAN [ZPIE17]. Pix2Pix uses conditional adversarial networks for image-to-image translation. Training is done with input and expected output picture pairs. CycleGAN does not require the images to be paired. The resulting generator network can be used for images that are similar to the input training data, producing an output that is similar to the set specified as expected.

3 OUR PROPOSAL IN DETAIL

Our goal is to accelerate particle-based fluid visualization in applications where perceptual correctness is important. First, we render metaballs using an inexpensive stochastic forward rendering solution. Then, we use neural network processing to either reconstruct a surface in image space, or render a scene with the reconstructed fluid surface featuring reflections and refractions. The neural networks are image-to-image GAN realizations (those of the Pix2Pix and CycleGAN network in particular) trained on image pairs produced by our stochastic renderer and our reference ray tracer.

3.1 Reference ray-casting and ray-tracing

In order to train the neural networks, we need expected output images for given sets of metaball particles. We created two types of such reference images: false-colored surface reconstructions encoding the surface normals (Figure 1, left), and recursively ray-traced fluid surface rendering with a surrounding environment (Figure 1, right). In both cases, we used ray-marching to find surface points intersected by primary or secondary rays. For primary rays, we used a screen-space A-buffer for accelerating the evaluation of the density field given by the metaball particles, but we used brute force for secondary rays. The rendering time of the reference images is still negligible compared to the time requirements of training the neural networks. In the ray-traced reference images, radiance along rays not hitting any surface was taken from an environment texture. Thus, the environment itself was encoded in the neural network, meaning that for different environments, different networks have to be trained — even if an existing network of another environment provides a good starting point for the training. This way, every pixel in the reference image contains the contribution of all possible combinations of reflections and refractions along the light paths.

Figure 1: Reconstructed surface target reference and ray traced target reference.

3.2 Stochastic rendering

We need a method for the fast rendering of the fluid particles that does not lose information about the particles occluded by other metaballs. This is particularly important for those particles that contribute to the primary surface, but surfaces further back may also be important, especially if refractions are also to be reconstructed. Horvath [Hor19] renders particles as circles, which accomplishes these goals, but offers a starting point for the neural network that is far from the expected output. We propose to use stochastic rendering, where we randomly discard fragments of billboards representing metaballs, false-colored using the camera-space sphere normals, and linear depth values. As shown in Figure 2, the stochastic renders are perceptually already quite close to the ray-cast reference, but with noise that has to be filtered. Part of the purpose of the neural network is to perform this filtering in a learned way to produce a smooth surface.

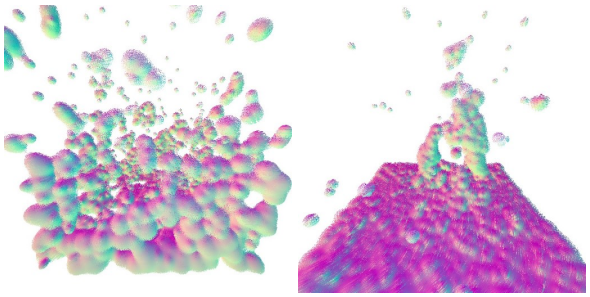

Figure 2: Images created using stochastic rendering. Images like these are inputs to both the training process, and to the image enhancement network used in real-time rendering.

3.3 Training datasets

To train the neural network, we render training sets consisting of image pairs. The *input* image is created using stochastic rendering, and the *target* image is rendered using either ray-casting or recursive ray tracing. Input images encode normals in camera space in their red and green channels, while camera-space linear depth is stored in the blue channel. To achieve stochastic rendering, we need to discard random fragments. For this

purpose, we used Cerisano's version [Cer15] of the *gold random* GPU-friendly fast random number generator. While this very simple code does not have high-quality PRNG properties, it is visually convincing, and serves our purpose of revealing occluded billboards.

In order to ensure realistic metaball configurations, we implemented SPH fluid simulation. In addition to physical forces, the simulation is controlled by animated triangle mesh models. In each simulation time step, particles are rendered with both the stochastic and the reference method to output images. The distance of the camera from the simulation space and the position of the camera changes randomly in each frame. Since camera depth is encoded in the blue channel, the farther the camera is from the metaball, the less blue there is in the color. Figure 3 shows an example image pair of the stochastically render input and the ray traced target output.

Figure 3: Image pair from the training set.

3.4 Training

We used a Tensorflow implementation of Pix2Pix [IZZE17] and a PyTorch implementation of CycleGAN [ZPIE17] to train the generator network with input and target image pairs. We do not give results separately for the two networks, as results were visually very similar, with CycleGAN performing faster. On the network obtained as the result, we ran the consecutive sequence of images saved from the application. The animation includes rotation and zooming around the simulation space.

3.5 Real-time rendering

The trained neural network can be used in a real-time application to render fluid surfaces. The input of the network is the stochastically rendered particle billboard cloud, and the output is presented by texturing it on a screen-filling quad.

If we use the surface reconstruction network, shading has to be performed in every pixel of the reconstructed surface. First, we decode the normal vector from the texture, then transforming it back to the world. Figure 4 shows the network output image and its shaded version using local illumination and environment mapping for approximated reflections and refractions.

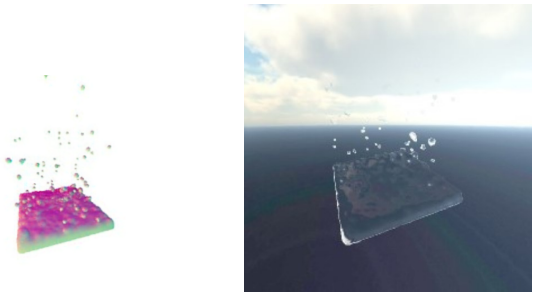

Figure 4: Surface reconstructed by the network and its shading with local illumination and environment mapping.

If we use the network trained with recursively ray-traced images, then further shading is not necessary.

4 RESULTS

We used a Tensorflow port of the Pix2Pix algorithm to train a deep neural network. We performed three trainings. First, we trained the network with 2078 image pairs, both of the pair in 512×512 resolution, showing incomplete and complete metaballs. The second one trained with input set contains metaballs shaded with surface reconstruction network and metaballs with ray-casting. The third trained with incomplete metaballs and metaballs with ray-casting. They performed around 30-50 epochs encompassing 72000-120000 training steps, taking approximately $8-15$ hours using an Nvidia Geforce RTX 2080 Super GPU. The same results could be achieved using the PyTorch implementation of CycleGAN in just 5 hours.

For testing we generated two test sets. One of them contains images of our fluid in consecutive simulation steps with changing camera setting. The other one contains zoom-in zoom-out pictures. Figure 5 and Figure 6 show the result of the first neural network. Therefore the environment not encoded in the neural network, but additional render pass is needed to shade the surface. Figure 7 shows the result of the second neural network. The environment is encoded and the network focus more on that part rather then the actual surface. The Figure 7 shows the result of the third neural network.

The Pix2Pix implementation proved to be inferior to the CycleGAN implementation both in terms of training time and network evaluation time. CycleGAN can be evaluated in $40ms$, which would be fast enough for actual real-time performance for full resolution targets.

5 LIMITATIONS AND FUTURE WORK

We plan to apply Wasserstein loss for the CycleGAN network, which has been proven to be useful in similar neural rendering applications [FIM18].

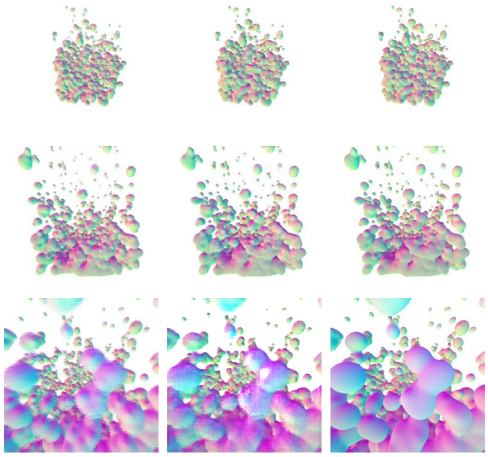

Figure 5: Stochastically rendered inputs (left), reconstructed surface images (middle), and target images (right).

Training the ray-tracing network with a fixed environment means that the method cannot be used for dynamic settings, including moving solid bodies mixed in the fluid. Such a static environment is acceptable in applications like computer games, where the game levels and rooms are designed in advance, and pre-training a fluid-lighting network for every relevant location would be reasonable. However, a solution using an environment map or scene capture on-the-fly would certainly extend the applicability of our approach. The most promising avenue would be training a network to render fluids and their lighting interactions with solid geometry and the environment in a single run.

Currently, the ray-tracing network does not only learn to shade the fluid surface, but also to create the background. This may diminish its ability to correctly render reflections and refractions. Modifying the training process to exclude backgrounds, and rendering the background in a forward manner would be desirable.

6 CONCLUSIONS

We have described a solution for enhancing fast fluid rendering with neural networks train in an adversarial manner. Our results indicate that the approach is viable, but further research is needed to extend the solution to more dynamic use cases and further improve performance.

7 ACKNOWLEDGEMENTS

This work has been supported by OTKA K-124124. The research presented in this paper, carried out by BME, was supported by the *Ministry of Innovation*, and the *National Research, Development and Innovation Office*, within the framework of the *Artificial Intelligence National Laboratory Programme*.

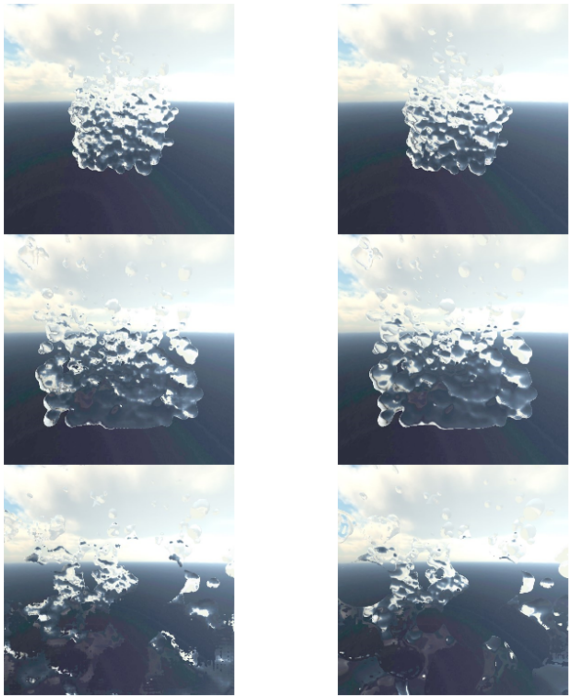

Figure 6: Final results of surface reconstruction with environment mapping (left) compared to the environment mapped reference surface (right)

8 REFERENCES

[BL88] David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[Bli82] James F Blinn. A generalization of algebraic surface drawing. *ACM transactions on graphics (TOG)*, 1(3):235–256, 1982.

[BLMD18] Giang Bui, Truc Le, Brittany Morago, and Ye Duan. Point-based rendering enhancement via deep learning. *The Visual Computer*, 34(6):829–841, 2018.

[CCB20] Ibtissam Constantin, Joseph Constantin, and André Bigand. On the use of deep active semi-supervised learning for fast rendering in global illumination. *Journal of Imaging*, 6(9):91, 2020.

[Cer15] Dominic Cerisano. Gold noise uniform random static. `http://https://www.shadertoy.com/view/ltB3zD`, 2015. Online; accessed 2021-03-15.

[FHR+21] Stanislav Frolov, Tobias Hinz, Federico Raue, Jörn Hees, and Andreas Dengel. Adversarial text-to-image synthesis: A re-

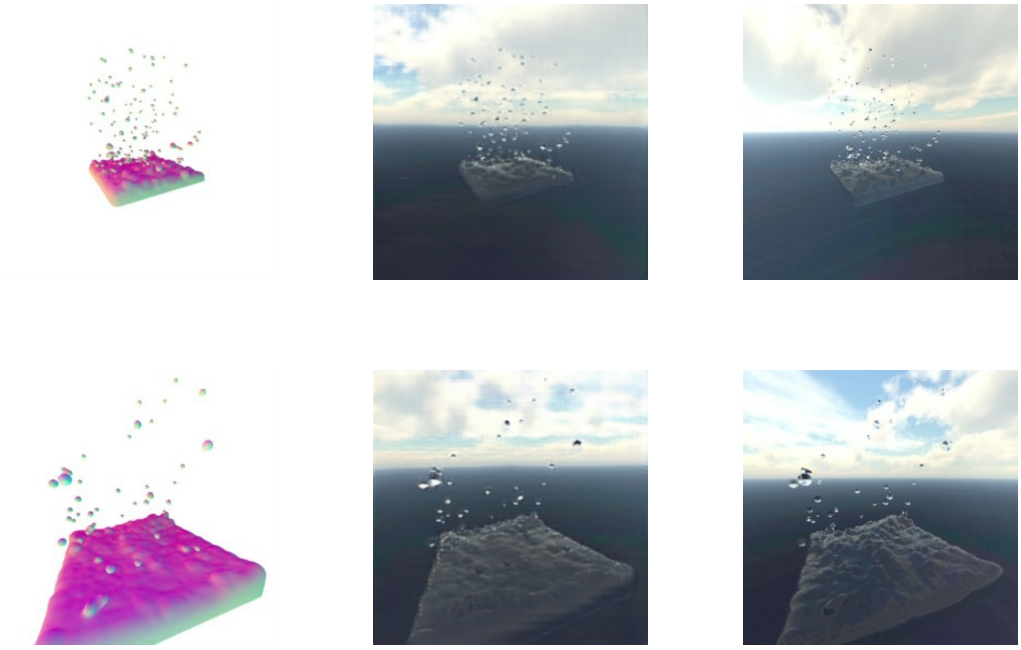

Figure 7: Surface reconstructed by the network (left), output images of the ray-tracing network (middle), and ray-traced reference (right).

view. *arXiv preprint arXiv:2101.09983*, 2021.

[FIM18] Anastasia Feygina, Dmitry I Ignatov, and Ilya Makarov. Realistic post-processing of rendered 3d scenes. In *ACM SIGGRAPH 2018 Posters*, pages 1–2. ACM New York, NY, USA, 2018.

[Hor19] Robert Horvath. *Image-space metaballs using deep learning*. PhD thesis, Wien, 2019.

[IDYN06] Kei Iwasaki, Yoshinori Dobashi, Fujichi Yoshimoto, and Tomoyuki Nishita. Real-time rendering of point based water surfaces. In *Computer Graphics International Conference*, pages 102–114. Springer, 2006.

[IZZE17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[KBT07] K KOOTEN, G BERGEN, and A TELEA. Point-based visualization of metaballs on a GPU. *GPU Gems*, 3, 2007.

[KSES12] Michael Krone, John E Stone, Thomas Ertl, and Klaus Schulten. Fast visualization of gaussian density surfaces for molecular dynamics and particle system trajectories. In *EuroVis (Short Papers)*, pages 067–071, 2012.

[KSN08] Yoshihiro Kanamori, Zoltan Szego, and Tomoyuki Nishita. GPU-based fast ray casting for a large number of metaballs. In *Computer Graphics Forum*, volume 27, pages 351–360. Wiley Online Library, 2008.

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[MGE07] Christoph Müller, Sebastian Grottel, and Thomas Ertl. Image-space GPU metaballs for time-dependent particle data sets. In *VMV*, pages 31–40, 2007.

[Nis85] Hitoshi Nishimura. Object modeling by distribution function and a method of image generation. *Trans Inst Electron Commun Eng Japan*, 68:718, 1985.

[SI12] László Szécsi and Dávid Illés. Real-time metaball ray casting with fragment lists. In *Eurographics (Short Papers)*, pages 93–96, 2012.

[TF17] Manu Mathew Thomas and Angus G Forbes. Deep illumination: Approximat-

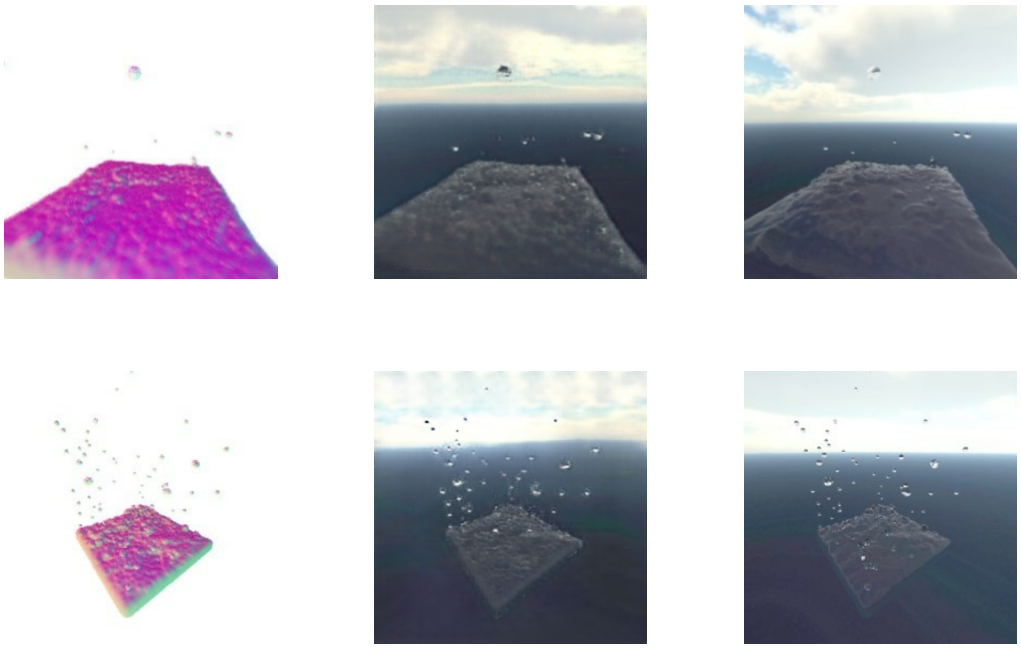

Figure 8: Stochastically rendered inputs (left), output images of the ray-tracing network (middle), and ray-traced reference (right).

ing dynamic global illumination with generative adversarial network. *arXiv preprint arXiv:1710.09834*, 2017.

[TFT+20] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020.

[vdLGS09] Wladimir J van der Laan, Simon Green, and Miguel Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 91–98, 2009.

[WK20] R Winchenbach and A Kolb. Multi-level memory structures for simulating and rendering smoothed particle hydrodynamics. In *Computer Graphics Forum*, volume 39, pages 527–541. Wiley Online Library, 2020.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The visual computer*, 2(4):227–234, 1986.

[XW16] Tian-Chen Xu and En-Hua Wu. View-space meta-ball approximation by depth-independent accumulative fields. In *SIGGRAPH ASIA 2016 Technical Briefs*, pages 1–4. ACM New York, NY, USA, 2016.

[ZHS18] Pourya Habib Zadeh, Reshad Hosseini, and Suvrit Sra. Deep-rbf networks revisited: Robust classification with rejection. *arXiv preprint arXiv:1812.03190*, 2018.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.