Water Simulation and Rendering from a Still Photograph

Ryusuke Sugimoto
University of Waterloo
Waterloo, ON, Canada
rsugimot@uwaterloo.ca

Mingming He
Netflix
Los Angeles, CA, USA
hmm.lillian@gmail.com

Jing Liao
City University
of Hong Kong
Kowloon, Hong Kong
jingliao@cityu.edu.hk

Pedro V. Sander
The Hong Kong University
of Science and Technology
Kowloon, Hong Kong
psander@cse.ust.hk

(a) Input   (b) Rendered   (c) Edited   (d) Msk.&Ref.

Figure 1: Given a single input image (a), our approach estimates the parameters, segmentation mask and reflection texture (d) needed to predict and render a realistic animated water surface (b), further enabling interactive editing with the water by placing synthetic objects (*e.g.* beach balls) on the surface (c). Input image: - Paul -/Flickr.

ABSTRACT

We propose an approach to simulate and render realistic water animation from a single still input photograph. We first segment the water surface, estimate rendering parameters, and compute water reflection textures with a combination of neural networks and traditional optimization techniques. Then we propose an image-based screen space local reflection model to render the water surface overlaid on the input image and generate real-time water animation. Our approach creates realistic results with no user intervention for a wide variety of natural scenes containing large bodies of water with different lighting and water surface conditions. Since our method provides a 3D representation of the water surface, it naturally enables direct editing of water parameters and also supports interactive applications like adding synthetic objects to the scene.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; *Image-based rendering*; *Computational photography*.

KEYWORDS

single-image animation generation, texture prediction, neural networks, optimization, screen-space reflection

1 INTRODUCTION AND RELATED WORK

Simulating and rendering water is an extensively studied problem in traditional computer graphics [Bridson and Müller-Fischer 2007; Darles et al. 2011]. It involves simulating the water dynamics using parametric models [Max 1981; Tessendorf 2001] and generating realistic shading that takes into account various optical properties of the water surface [Tessendorf 2001]. Empirical parametric models can generate visually appealing water geometry and appearance and allow the result to extend arbitrarily in space and time. However, they usually rely on a large number of parameters to model the water surface dynamics (*e.g.* wind speed, wind direction, *etc.*) and lighting conditions. This often requires much time and manual effort to tune the parameters to achieve the desirable effect.

In this work, we are interested in automatically animating the water surface in a still photograph. In particular, we seek to generate and render the high-fidelity motion of large water bodies such as lakes and seas in a fully automatic manner. To avoid crossing into the "uncanny valley", previous methods either require significant user annotation to specify the water motion in the image [Chuang et al. 2005; Le et al. 2022; Okabe et al. 2018], such as the dynamic region boundary and flow directions in the concurrent work of Le et al. [2022], or assume the availability of video examples or multiple frames containing the target water motion [Gui et al. 2012; Lin et al. 2007; Okabe et al. 2009, 2011; Prashnani et al. 2017; Sun et al. 2003] and use such inputs to drive image animation. The video-driven methods by Gui et al. [2012]; Okabe et al. [2009, 2011]; Sun et al. [2003] extract the motion field (*e.g.* optical flow) from a driving video and transfer motion into the image, allowing the
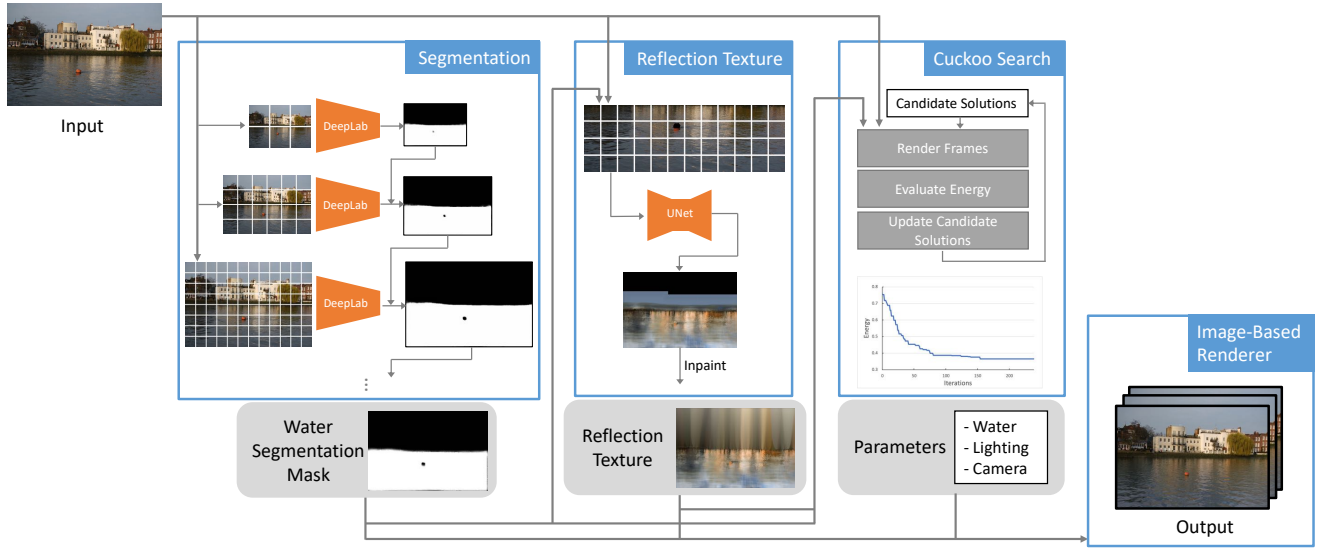
Figure 2: System overview. Given an input image, we first segment water, then predict reflection texture from the water segment, and estimate other parameters. With the reflection texture and parameters, we generate the water animation using an image-based renderer.

still water to come alive and mimic the motion in the driving video. To avoid the instability of optical flow, Prashnani et al. [2017] propose to use phase variations to model the motion. However, these methods cannot decouple appearance and dynamics in the target image, which limits their capacity in processing the water surface with distorted fluctuation and reflectance. The recent works [Endo et al. 2019; Tesfaldet et al. 2018] decouple appearance and dynamics using two pre-trained convolutional neural networks (CNNs). Tesfaldet et al. [2018] propose to transfer dynamics only from the driving video while preserving the appearance in the target image. The work by Endo et al. [2019] predicts optical flows to tackle the single-image animation generation task. Since these methods use spatially-invariant statistics to represent the appearance, they are also limited to processing spatiotemporally homogeneous data. Recently, another impressive work [Holynski et al. 2021] proposes a motion representation based on Euler integration and synthesizes plausible motion for a given image by learning from a large-scale video dataset. These non-parametric methods synthesize dynamic water using a pure generation model, much like a black box, without building physically correct geometry and reflectance, thus limiting the resulting quality and resolution and also resulting in a lack of control regarding the appearance, diversity, and consistency of the animated water.

Taking only a *single* image of a target scene, we aim to animate and render the water regions. Our work lies in the confluence of traditional parametric models and learning-based non-parametric models of water generation in an attempt to combine the best of both worlds: the artifact-free rendering and flexible control of parametric models and the generalization ability of non-parametric models. We represent the water geometry and appearance using an empirical parametric model [Tessendorf 2001], and automatically estimate the model parameters from the input image using

both optimization-based and learning-based methods. With the estimated parameters, we can simulate and render the water that is visually similar to the input photograph and directly use the parametric model for creation and editing of the water animation.

More specifically, the parameter estimation is the core of our method. It is a challenging ill-posed problem since we need a full set of parameters from a single input image, including numerical parameters of appearance (*i.e.* color), dynamics (*i.e.* wind speed, wind direction, and wave choppiness), cameras (*i.e.* angle, height, and field of view) and environmental lighting (*i.e.* spherical harmonics (SH) [Ramamoorthi and Hanrahan 2001]), as well as the reflectance of the entire scene. To address these challenges, we first formulate the reflectance estimation as an image synthesis task of generating a reflection texture from the water image using deep neural networks. Similar methods have achieved great success in many related image-based prediction tasks from a single image, such as distortion correction [Li et al. 2019a,b], reflection removal [Li et al. 2020; Wen et al. 2019], and denoising [Guo et al. 2019; Wei et al. 2020]. Due to practical constraints, we synthesize water images with realistic reflection effects for the supervised learning. Next, to predict lighting and other water dynamics parameters, we propose an adaptation of the cuckoo search metaheuristic [Yang and Deb 2009], due to its simplicity and flexibility in exploring different candidate solutions.

We combine these techniques of parameter estimation together into a novel system to animate the water surface in a still photograph, as shown in Fig. 2. We first develop a progressive framework for water segmentation of a high-resolution image (Sec. 2). We then leverage a supervised learning method to predict reflectance information for the water surface as needed by our real-time renderer (Sec. 3). Next, we use the cuckoo search meta-heuristic to estimate water surface parameters with a parallelized energy evaluation

(a) Input   **(b) Level 1**   **(c) Level 2**   **(d) Level 3**
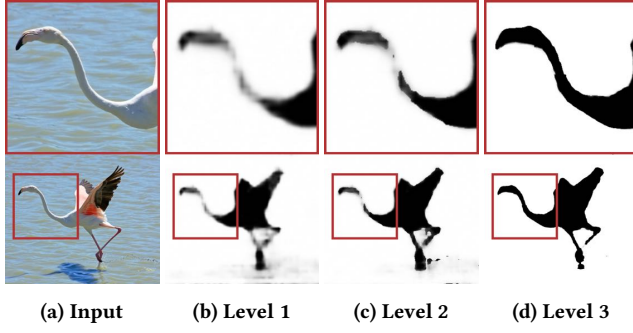
Figure 3: Intermediate results of the progressive patch-based segmentation framework, As the image level and resolution increase, fine-detailed segmentation edges are progressively detected. Input image: Andrea S/Flickr.

scheme (Sec. 4). Equipped with the predicted reflectance and the estimated parameters, we customize a real-time renderer with an efficient image-based screen space local reflection method (Sec. 5) to generate the final animation. We demonstrate the effectiveness of our method on a variety of scenes and interactive applications.

2 WATER SEGMENTATION

Semantic image segmentation is one of the fundamental topics in computer vision and recently has been solved effectively using deep convolutional neural networks deployed in a fully convolutional manner [Chen et al. 2018a, 2017, 2018b; Liu et al. 2019; Long et al. 2015; Zhao et al. 2018, 2017]. Although the state-of-the-art systems propose to segment images at high resolution (usually $1K$ or $2K$) by either incorporating multi-resolution branches [Zhao et al. 2018] or forming a hierarchical architecture search space in the network [Liu et al. 2019], their scalability is still limited by computing resources.

To segment a water image at higher resolution (*e.g.* $4K$, $8K$ or higher in our application setting, we propose a progressive patch-based segmentation framework. We first build an image pyramid of multiple levels of detail by iteratively downscaling the input image to a resolution with the longest edge no large than 512 pixels at the lowest level. At each level, we dice the image into a set of patches (512×512 pixels) in a 2D grid structure with 50% overlap between adjacent patches. Then we start segmenting water on each patch from the lowest level using a pre-trained model of water segmentation, bilinearly upsampling the segmentation probability to the next level while iteratively updating the patches with pixel error larger than a threshold (we set it to 0.2 in our implementation). For the pixels covered by multiple levels or multiple overlapping patches, we take the maximum probability as output. Our motivation is that low-level segmentation provides global information while the high-level segmentation ensures higher accuracy around boundaries. The iterative refinement repeats as the level increases until the image resolution reaches $4K$. Then for each patch, we upscale its predicted mask to the original resolution using the guided filter [He and Sun 2015] to further refine the often over-smoothed segmentation result. Fig. 3 shows the segmentation result of each level and it can be seen that the segmentation boundary is clearer at higher levels. The water patch segmentation network is based

(a) Input   **(b) Ground truth**   **(c) Predicted**   **(d) Rendered**
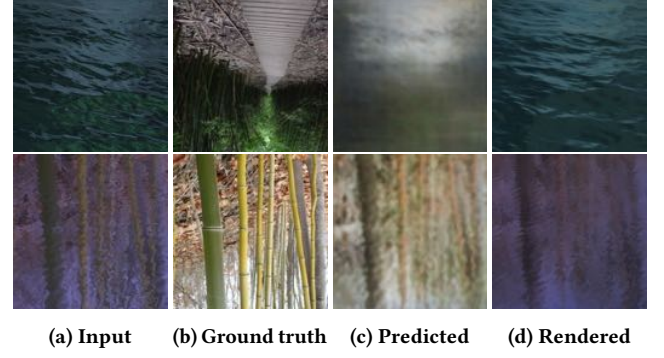
Figure 4: Images from the dataset along with their resulting predictions. The first row shows an example with turbulent water and a blurry reflection. The second row shows an example where the water is calm and the predicted texture is sharper. (d) shows the patches rendered using the predicted reflection textures (c). Blurry prediction suffices when the water is turbulent. Ground truth: Places [Zhou et al. 2014].

on the advanced architecture of Deeplab [Chen et al. 2018a]. For further details on the training, please refer to the supplemental material.

3 REFLECTION TEXTURE GENERATION

Next, we aim to generate a *reflection texture*. This texture will later be used during shading of the the water in order to more accurately model reflections on the water surface. The rendering algorithm is described in Sec. 5.

When the water surface is as flat as a mirror, our approach seeks to generate a sharp reflection texture, preserving all high frequency details of the scene in the water reflection. In contrast, when the water surface is turbulent, the approach generates a blurry texture by resulting in a more realistic approximation to the complex interaction between light and turbulent wave dynamics.

The texture is built on image space and we seek to store the reflected color of the water surface as if the water was a completely flat mirror. The colors are then dilated to the regions beyond the water surface using an inpainting technique. Due to the high resolution, overlapping texture patches are first predicted by a patch-based learning network and then stitched together into a full texture map.

Dataset. . Our dataset consists of pairs of an image patch of water and its corresponding reflection texture patch. Since the ground truth of reflection texture is not available, we create a synthetic dataset. We use an image patch (as a reflection texture) and random parameters to render a water image patch with a size 224×224. Then, the rendered water image patch and its reflection texture patch are treated as a ground truth pair for training (see Fig. 4). Further details on the dataset can be found in the supplemental material.

Network architecture. . Our network learns a mapping from the rendered image patch to its reflection texture patch. Despite the reduced rendering model and difference between the synthetic images and real images, this method allows us to generate a reflection

(a) Input   (b) Our method   (c) Neural network
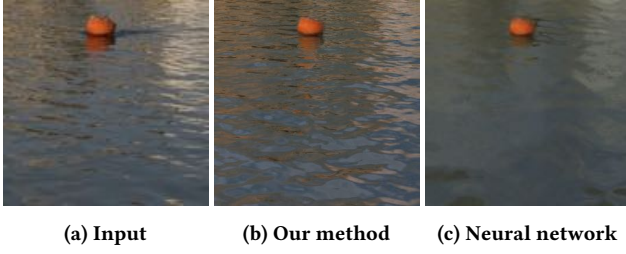
Figure 5: Comparison between our method and a neural network approach. The result of the neural network method in this example lacks the high frequency waves present in the input image. Furthermore, the overall color is shifted towards green.

(a) Input   (b) Our method   (c) w/o color metric

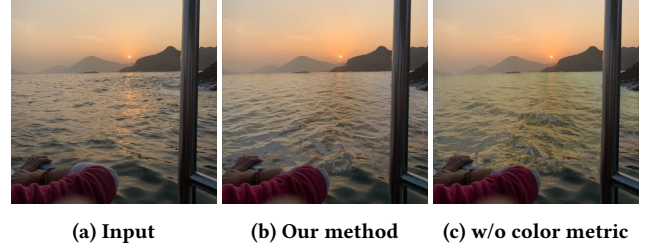**Figure 6: Comparison between our method with and without the color similarity metric.**

texture from any image with a water surface. The network architecture is based on UNet [Ronneberger et al. 2015] with residual blocks [He et al. 2016] and multi-level skip connections between the encoder and decoder that can preserve sharp edges in reflection textures when desired. Note that the blurred reflection texture that is generated for turbulent water does not often degrade the quality of rendering when used, since they will be applied using turbulent water parameters. Refer to the supplemental material for further details on the network architecture.

Loss evaluation. . During training, we distinguish water and non-water regions using a random mask image m drawn from a set of mask images and their inverses generated using the ground truth annotation of the COCO dataset [Lin et al. 2014]. We apply the mask to the input patch (i.e. multiply the mask pixel value to the input image pixel at each pixel location) and feed it to the network to get a prediction patch. Notice that the range of mask value is [0, 1]. This is to simulate real input patches, which may have both water and non-water regions. Then, the loss is defined as

$$L(x, y) = \frac{1}{N} \sum_{i}^{N} (m_i + \lambda(1 - m_i)) |x_i - y_i|, \tag{1}$$

where N is the number of pixel locations, x_i, y_i represent the pixel color of the two patches x and y at the pixel location i respectively, and m_i is the mask value at that pixel location. We let $\lambda = 0.1$ to impose a small loss on pixels that are outside of water regions. We do not expect the network to fill non-water regions with accurate reflection colors; we need the network to fill such regions with colors that transition smoothly from the neighbouring water regions.

Stitching and inpainting. During testing, we partition the input image and the mask into patches with 80% overlap in each dimension and feed each image patch into our network to get its corresponding reflection texture patch. These reflection texture patches are stitched together using weights following a Gaussian kernel within the overlap region to ensure a smooth transition. After stitching, we inpaint the non-water regions of the full reflection images using the method of [Telea 2004]. This is needed since we may occasionally need to fetch the reflection color slightly outside of the water region in cases where the water surface is turbulent.

4 PARAMETER ESTIMATION BY CUCKOO SEARCH

In addition to the segmentation mask and the reflection texture presented in the previous sections, the rendering algorithm takes a 21-dimensional vector of wave, wind, camera, and lighting parameters. The supplemental material contains a table with all of the parameters and their respective ranges.

We initially tried to design a neural network similar to the reflection texture generation network that performed parameter inference by creating a synthetic dataset that included ground truth parameters. However, the predicted parameters did not produce plausible results when used for rendering, as shown in Fig. 5. The problem is ill-posed as different sets of parameters may give a similar output (e.g. high wind speed with a high camera position vs. low wind speed with a low camera position). We believe that this ambiguity makes it challenging for the network to learn the parameters accurately. We investigated different techniques to explore the space of possible parameter solutions including traditional optimization methods and learning-based approaches. Finally, we have found an adaption of the cuckoo search metaheuristic [Yang and Deb 2009] to be a suitable choice for our ill-conditioned and non-differentiable problem. This is because the cuckoo search does not assume any specific characteristics of the optimization problem such as convexity and does not require a gradient of the solution.

In our application, we use cuckoo search with an energy function based on a combination of the DISTS similarity metric [Ding et al. 2020] and an HSV color histogram metric. For each candidate parameter set, we render an image and calculate a distance between rendered image y and the original input image x. We find a parameter set that minimizes this distance.

4.1 Energy Function

As mentioned above, our energy function considers both the DISTS metric and a color histogram metric:

$$E_T + \lambda E_C, \tag{2}$$

where E_T is DISTS energy, E_C is the color dissimilarity energy, and λ regulates the tradeoff ($\lambda = 1.0$ in our implementation).

Texture similarity using DISTS index. Since the water surface is dynamically changing, we need a distance metric that is not sensitive to local variations yet globally consistent with human perceptual scores. We have found that the DISTS index is suitable

for this task and apply it directly in our approach:

$$E_T = d(x, y) , \qquad (3)$$

where $d(x, y)$ is the DISTS index measuring the dissimilarity between images x and y.

Color similarity using HSV color histogram. While the DISTS metric achieves good results in measuring similarity in the overall structure of the image content, it has not been as effective in ensuring color similarity. Thus we added a color histogram distance metric, which measures similarity in HSV color space. More specifically, the images are converted to HSV color space, and then for each image, all pixels are classified into one of $24 \times 8 \times 8$ partitions based on hue (24 classes), saturation (8), and value (8). Note that we allocate more partition resolution to the hue. The distance between two sets of bins is then measured using the Hellinger distance, which is an effective technique to measure the amount of overlap between two distributions:

$$E_C = H(x, y) = \sqrt{1 - \frac{1}{\sqrt{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}} \sum_{i=1}^{n} \sqrt{x_i y_i}} , \qquad (4)$$

where n is the number of partitions, and x_i, y_i represent the number of pixels that fall into partition i from images x and y, respectively. In Fig. 6, we compare our final method with and without considering the HSV color similarity metric. Note that the addition of the color similarity results in renderings that are much more consistent with the colors of the input image.

Evaluation details. The input to our evaluation is a set of two images: the original input image and an image rendered with a candidate parameter set. First, we crop the bounding box for all water regions for both the original image and reflection texture and resize them to 256x256. Then, using the resized reflection texture and the candidate parameter set, an image of the same resolution is rendered. The rendered image and the resized original image are passed to the two metrics for evaluation.

4.2 Algorithm

A simple cuckoo search maintains a set of n nests (25 in our implementation), each with a candidate solution, or egg (*i.e.*, values for each of the parameters). In each iteration, a new solution, or cuckoo egg, is generated for each nest via a Lévy flight, a random walk in the parameter space in which the step size follows the Lévy distribution. The cuckoo egg replaces an egg in another nest if it improves upon the latter. Next, for each nest, a new cuckoo egg is generated by mutation and replaces the egg in the nest if it is of improved quality. At the end of each iteration, the algorithm keeps the best nests and drops a small fraction k of the nests (5 in our implementation) replacing them with new random eggs. The supplemental material describes a parallelized version of the algorithm for improved efficiency and provides the pseudocode for both.

Termination condition. The algorithm terminates when the optimization no longer yields significant improvements to the best egg. Let the energy of the best egg at the end of iteration k be $E(k)$. To reduce the influence of random oscillations, we first apply a

(a) ray marching   **(b) texture sampling**
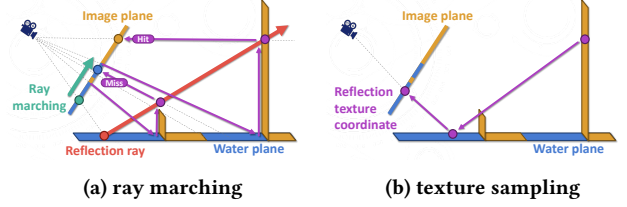
Figure 7: The image-based reflection method. The objects are mapped onto the water plane to determine the reflection source point in 3D. We perform ray-marching in image space with additional checks in 3D space to consider the height of wall proxy objects. Then, we fetch the reflection color embedded in the water surface from the reflection texture.

smoothing filter to $E(\cdot)$, yielding the smoothed energy $E'(\cdot)$:

$$E'(k) = \sum_{i=k-s+1}^{k} \{i - (k - s)\} E(i) , \qquad (5)$$

where s is the filter size. When $\{E'(k) - E'(k-1)\}/E'(k) < \epsilon$, the algorithm terminates. We use $s = 100$ for smoothing and $\epsilon = 0.0001$ as a conservative enough threshold.

5 RENDERER

Our algorithm renders realistic deep water bodies with local reflections, taking as input the the segmentation mask and reflection texture as well as the wave, wind, camera, and lighting parameters mentioned earlier and listed in the supplemental material.

Realistic simulation of water has been researched extensively in the graphics community. The methods can be classified into two main categories: physically-based models and empirical methods [Darles et al. 2011]. Physically-based methods are suitable for shallow water simulation and are capable of simulating full three dimensional behavior of water [Bridson and Müller-Fischer 2007]. Empirical methods generate 2D displacement maps of water surface of deep water scenes. Empirical methods can be further categorized into spatial-domain methods, spectral-domain methods, and hybrid methods [Darles et al. 2011]. In our work, we employ the spectral domain approach presented in Tessendorf [2001] to generate a displacement map of water surface using the wave and wind parameters. This is a commonly used approach in film production and real-time applications.

Our approach considers the fact that the surface of the water may extend infinitely and also allows the user to interactively zoom. Thus the naïve approach of constructing a large and dense mesh is not practical. We instead adopt the projected grid LOD method from Johanson and Lejdfors [2004]. More specifically, in the vertex shader, we generate a screen space mesh. We project each vertex to the water surface plane and apply the displacement based on the computed displacement map before projecting back to screen space. This results in a nearly uniformly and dense distribution of mesh vertices in screen space.

Image-based reflection. Water surface reflection is an essential component in making the water surface appear realistic. Tessendorf [2001] provides a good summary on surface wave optics. While

common implementations to achieve this goal use environment mapping, it is challenging to estimate an environment map from one static image and apply it to the entire scene. Furthermore, environment maps have a limitation of not being able to simulate local reflections. We take a similar approach to the screen space reflection method of McGuire and Mara [2014]. This is a practical method used for real-time applications which computes reflection given the normal and depth information from the scene. In our application, we do not have this information directly, but we can approximate it given the camera pose and water mask. Also, rather than retrieving the reflection color from the surface of objects, we make use of the reflection component embedded within the water regions stored in the precomputed reflection texture to allow reflection estimations even when objects are absent in the input image.

We first place *vertical curved walls* along water boundaries in world coordinates as proxies for 3D objects. The exact positions of the walls are automatically determined by projecting the water mask on the image plane onto the ground-level plane using the estimated camera parameters (see Fig. 7 for a diagram with a simple example). For each pixel, we compute the collision between the reflection ray from water facet and a vertical wall in world space. We compute the texel coordinate of the reflection texture which contains the corresponding reflection color. If the reflection ray completely misses the vertical walls, we retrieve a color from the closest point in the reflection texture. This approximation does not affect the quality of the final image significantly. This follows from the fact that there is very little contribution from the reflection when the incident angle is small. Since that is usually the case when the reflection vector does not hit any of the curved planes, the color of the water facet is dominated by the refraction term.

More specifically, these are the steps to compute the reflection color as implemented in our fragment shader. Further optimizations to improve running time are discussed in the supplemental material.

(1) *Calculate the reflection vector.* Given the view and normal vectors, we compute the reflection vector in world space and project it onto screen space.
(2) *Compute wall collision points.* We then apply ray marching in screen space to find the collision points with water boundaries. Note that there can be multiple collisions per ray as shown in Fig. 7a. We project the collision points back to world coordinates such that the points now reside on the water boundaries in world coordinates (forward purple arrows). We then calculate the collision points of the reflection vector and the object using the direction of the original reflection vector.
(3) *Determine first valid collision point.* If a collision point lies on the vertical wall, the collision point is a valid reflection color source point. To determine whether the wall is high enough to result in a collision, we use an approximation that projects the collision point back to screen space and checks the mask value of that point to ensure it is not on the water surface. Then, we choose the valid collision point that is closest to the ray marching starting point.
(4) *Calculate the reflection texture coordinates.* We then calculate the point on the water surface plane that contains the reflection

color under a flat mirror assumption (Fig. 7b). Finally, we sample the reflection texture at that position to retrieve the color.

6 EXPERIMENTS

We test our system on a dataset consisting of 67 images (32 from Places [Zhou et al. 2014] and 35 in-the-wild images) with a variety of water scenes including oceans, rivers, ponds, and lakes. The image resolutions ranged from 748×421 to 4032×3024. All experiments are performed on a PC with an 8 core Ryzen 2700X CPU, 16GB of RAM, and an NVIDIA GeForce RTX 2070 GPU. Please refer to the accompanying video for animated renderings and additional results.

Runtime. The runtime efficiency of our approach is proportional to the number of pixels in the water regions of the input image. For a 4K image (4032×3024) with the water occupying approximately one-third of the image, performing the water segmentation takes approximately 7 seconds, predicting the reflection texture takes 9 seconds, and estimating the parameters using the cuckoo search metaheuristic takes 4.5 minutes to evaluate about 19000 different candidate solutions. About 60% of execution time for optimization is spent on rendering, 30% on the DISTS metric evaluation, and 10% on color metric evaluation. The execution time for the optimization is indeed independent of input image size, as we evaluate the energy using a fixed-size image patch as described in Sec. 4.1. A theoretical convergence analysis of the cuckoo search is out of scope for this paper and is still an open problem except for a simplified version of the algorithm [He et al. 2018]. We implement a real-time renderer using WebGL for users to view and interact with the resulting animation. Our optimized renderer reaches 50-60fps at 4K resolution.

Results. We demonstrate the final results on a variety of input images generated by our method in Figs. 1 and 8. We also visualize some close-up details, the predicted segmentation mask, and the reflection texture. Note the high accuracy of the mask, particularly for large water bodies of varying shapes. Our reflection texture network is able to generate a high-quality reflection texture for images with both calm and turbulent waters. Naturally, for calm waters, the reconstructed reflection is much sharper than for turbulent waters as shown in the third case of Fig. 8. Recall that in most cases, to render the turbulent water, a smooth reflection texture is sufficient since reflection details are not discernible in a turbulent setting.

User study. In addition to the visual demonstration, we conduct a user study to evaluate the realism of our composited results. We randomly select 30 images from the testing dataset and create a poll of 30 "real or rendered" questions for each user. For every question, the user is randomly shown either the real input or the rendered result and then asked whether the image looks realistic or not. We retrieve 2,160 answers from 72 users and compute the "real rate", which is the percentage of images that the users tag as real. The real rate for real images is 81.67% while the real rate for rendered images is 65.46%, indicating that most of our generated results are thought to be as realistic as real images. As shown in this user study, although our method can hardly ensure pixel-level accuracy with respect to the ground truth image, it is able to
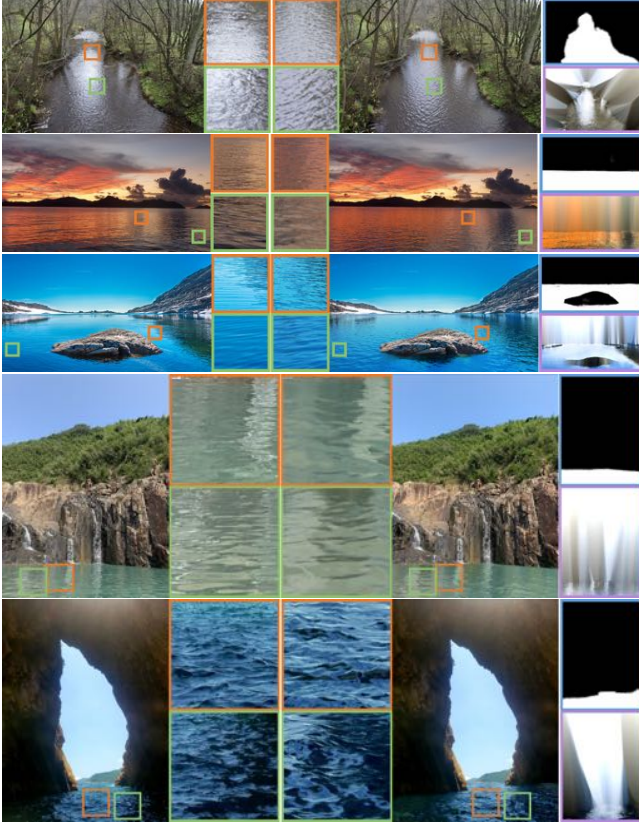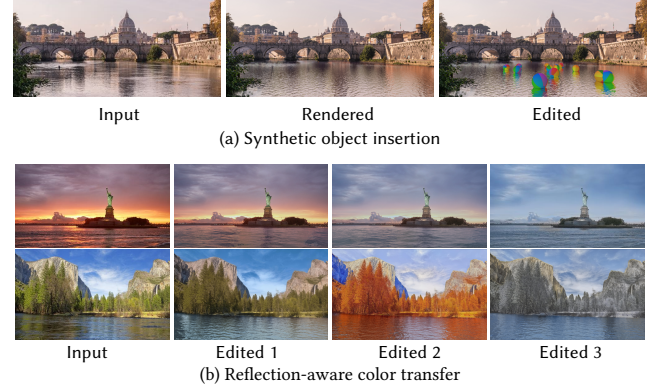
Figure 8: Rendered results and closeups for different input scenes, along with the predicted segmentation masks and reflection textures. The input images are shown in the first column. Input images: Edward Nicholl/Flickr (first), Foliez/Pixabay (second), and Unknown/PxHere (third).

(a) Synthetic object insertion

(b) Reflection-aware color transfer

Figure 9: Applications. In (a), our method allows to place synthetic objects on the water surface and generates realistic reflections. In (b), given an input image, we transfer its color to simulate different environments and predict corresponding reflection textures. By interpolating between the predicted parameters, the edited results vary smoothly. Input Images: (a) Krzysztof Golik/Wikimedia (b) Confaulk/Wikimedia (top) and Rixie/Adobe Stock (bottom).
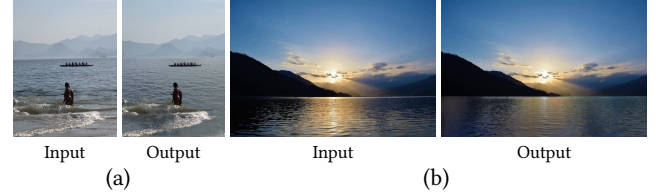
Figure 10: Limitations. In (a), our method fails to capture the dynamics of breaking waves. In (b), our method does not handle strong reflection of sunlight. The second input image: Eshan Chandra/Pixabay.

generate photorealistic results with high fidelity at a low rendering cost for real-time applications.

Applications. Our method generates 3D representations of the water surface. It not only allows direct editing of parameters to control wave, wind, and lighting but also enables interesting applications such as insertion of synthetic objects and reflection-aware color transfer as shown in Fig. 9. To render the synthetic objects into the water (Fig. 9a), we first use the estimated camera pose and wave parameters to compute reflection and refraction vectors in world coordinates. Then, we compute intersections between the rays and the synthetic objects. The objects are shaded based on their normals and our estimated SH coefficients. In the reflection-aware color transfer (Fig. 9b), we can simulate consistent water animation as the color of the surrounding environment changes producing interesting time-lapse videos as shown in the supplemental video. Given an input image, we first apply a color transfer method [He et al. 2019] to simulate its appearance at different times of the day or seasons. Taking the original image and one or more of its recolored results as input, we separately estimate the parameters and textures for each input image. We then use the median

values among all the estimates for the wave dynamics and camera pose parameters, which circumvents artifacts due to any minor differences in the estimated parameters among all input images. Finally, we linearly interpolate the reflection textures and lighting parameters to generate a water animation with smoothly varying lighting conditions.

7 LIMITATIONS AND SUMMARY

While our method works on a wide variety of scenes, there are situations where our approach is not applicable due to the inherent limitations of the water surface simulation model employed. For instance, our method cannot simulate flowing water such as waterfalls and dynamic water breaks such as ocean waves on the coastal line (Fig. 10a). We also do not propose a detailed treatment of refraction but instead use a single color in place of the refraction component, which is a valid assumption case of deep water. In addition, our lighting model does not handle strong reflection of sunlight in some cases (Fig. 10b) because both input and output of the reflection texture prediction are standard 24-bit RGB images, which cannot store high radiance by strong lights. Furthermore, we

assume that there is one set of parameters for the body of water, and this assumption may not hold very close to the coastline. Our current implementation does not support multiple bodies of water in the same scene, but this can be easily addressed with a trivial extension.

In summary, we present a technique that renders realistic animated water surfaces given a single still input photograph. Our approach determines rendering parameters and water reflection textures using a combination of neural networks and optimization techniques. The results are then fed to our renderer which displays the animated water in real-time. The entire process is fully automatic and relies on a single input image. Our approach generates realistic results for a wide variety of natural scenes with different lighting and water surface conditions, yielding particularly good results for deepwater scenes. The generated 3D scenes show the potential to support a variety of interactive applications.

ACKNOWLEDGMENTS

REFERENCES

Robert Bridson and Matthias Müller-Fischer. 2007. Fluid Simulation: SIGGRAPH 2007 Course NotesVideo Files Associated with This Course Are Available from the Citation Page. In *ACM SIGGRAPH 2007 Courses* (San Diego, California) *(SIGGRAPH '07)*. 1–81.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2018a. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.* 40, 4 (2018), 834–848.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking Atrous Convolution for Semantic Image Segmentation. *CoRR* abs/1706.05587 (2017).

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. 2018b. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII (Lecture Notes in Computer Science, Vol. 11211)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 833–851.

Yung-Yu Chuang, Dan B. Goldman, Ke Colin Zheng, Brian Curless, David Salesin, and Richard Szeliski. 2005. Animating pictures with stochastic motion textures. *ACM Trans. Graph.* 24, 3 (2005), 853–860.

E. Darles, B. Crespin, D. Ghazanfarpour, and J.C. Gonzato. 2011. A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics. *Computer Graphics Forum* 30, 1 (2011), 43–60.

Keyan Ding, Kede Ma, Shiqi Wang, and Eero P. Simoncelli. 2020. Image Quality Assessment: Unifying Structure and Texture Similarity. *CoRR* abs/2004.07728 (2020).

Yuki Endo, Yoshihiro Kanamori, and Shigeru Kuriyama. 2019. Animating landscape: self-supervised learning of decoupled motion and appearance for single-image video synthesis. *ACM Trans. Graph.* 38, 6 (2019), 175:1–175:19.

Yan Gui, Lizhuang Ma, Chao Yin, and Zhihua Chen. 2012. Preserving global features of fluid animation from a single image using video examples. *J. Zhejiang Univ. Sci. C* 13, 7 (2012), 510–519.

Shi Guo, Zifei Yan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2019. Toward Convolutional Blind Denoising of Real Photographs. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 1712–1722.

Kaiming He and Jian Sun. 2015. Fast Guided Filter. *CoRR* abs/1505.00996 (2015).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 770–778.

Mingming He, Jing Liao, Dongdong Chen, Lu Yuan, and Pedro V. Sander. 2019. Progressive Color Transfer With Dense Semantic Correspondences. *ACM Trans. Graph.* 38, 2 (2019), 13:1–13:18.

Xing-Shi He, Fan Wang, Yan Wang, and Xin-She Yang. 2018. *Global Convergence Analysis of Cuckoo Search Using Markov Theory*. Springer International Publishing, Cham, 53–67. https://doi.org/10.1007/978-3-319-67669-2_3

Aleksander Holynski, Brian L. Curless, Steven M. Seitz, and Richard Szeliski. 2021. Animating Pictures With Eulerian Motion Fields. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 5810–5819.

Claes Johanson and Calle Lejdfors. 2004. Real-time water rendering. *Lund University* (2004).

Thi-Ngoc-Hanh Le, Chih-Kuo Yeh, Ying-Chi Lin, and Tong-Yee Lee. 2022. Animating still natural images using warping. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* (2022).

Chao Li, Yixiao Yang, Kun He, Stephen Lin, and John E. Hopcroft. 2020. Single Image Reflection Removal Through Cascaded Refinement. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 3562–3571.

Xiaoyu Li, Bo Zhang, Jing Liao, and Pedro V. Sander. 2019a. Document rectification and illumination correction using a patch-based CNN. *ACM Trans. Graph.* 38, 6 (2019), 168:1–168:11.

Xiaoyu Li, Bo Zhang, Pedro V. Sander, and Jing Liao. 2019b. Blind Geometric Distortion Correction on Images Through Deep Learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 4855–4864.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V (Lecture Notes in Computer Science, Vol. 8693)*, David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer, 740–755.

Zhouchen Lin, Lifeng Wang, Yunbo Wang, Sing Bing Kang, and Tian Fang. 2007. High Resolution Animated Scenes from Stills. *IEEE Trans. Vis. Comput. Graph.* 13, 3 (2007), 562–568.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Fei-Fei Li. 2019. Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 82–92.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 3431–3440.

Nelson L. Max. 1981. Vectorized procedural models for natural terrain: Waves and islands in the sunset. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1981, Dallas, Texas, USA, August 3-7, 1981*, Doug Green, Tony Lucido, and Henry Fuchs (Eds.). ACM, 317–324.

Morgan McGuire and Michael Mara. 2014. Efficient GPU Screen-Space Ray Tracing. *Journal of Computer Graphics Techniques (JCGT)* 3, 4 (9 December 2014), 73–85.

Makoto Okabe, Ken-ichi Anjyo, Takeo Igarashi, and Hans-Peter Seidel. 2009. Animating Pictures of Fluid using Video Examples. *Comput. Graph. Forum* 28, 2 (2009), 677–686.

Makoto Okabe, Ken Anjyo, and Rikio Onai. 2011. Creating Fluid Animation from a Single Image using Video Database. *Comput. Graph. Forum* 30, 7 (2011), 1973–1982.

Makoto Okabe, Yoshinori Dobashi, and Ken Anjyo. 2018. Animating pictures of water scenes using video retrieval. *Vis. Comput.* 34, 3 (2018), 347–358.

Ekta Prashnani, Maneli Noorkami, Daniel Vaquero, and Pradeep Sen. 2017. A Phase-Based Approach for Animating Images Using Video Examples. *Comput. Graph. Forum* 36, 6 (2017), 303–311.

Ravi Ramamoorthi and Pat Hanrahan. 2001. An efficient representation for irradiance environment maps. In *SIGGRAPH 2001*. 497–500.

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 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 9351)*, Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi (Eds.). Springer, 234–241.

Meng Sun, Allan D. Jepson, and Eugene Fiume. 2003. Video Input Driven Animation (VIDA). In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*. IEEE Computer Society, 96–103.

Alexandru Telea. 2004. An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics Tools* 9 (01 2004). https://doi.org/10.1080/10867651.2004.10487596

Matthew Tesfaldet, Marcus A. Brubaker, and Konstantinos G. Derpanis. 2018. Two-Stream Convolutional Networks for Dynamic Texture Synthesis. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 6703–6712.

Jerry Tessendorf. 2001. Simulating Ocean Water. *SIGGRAPH'99 Course Note* (01 2001).

Kaixuan Wei, Ying Fu, Jiaolong Yang, and Hua Huang. 2020. A Physics-Based Noise Formation Model for Extreme Low-Light Raw Denoising. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020.* IEEE, 2755–2764.

Qiang Wen, Yinjie Tan, Jing Qin, Wenxi Liu, Guoqiang Han, and Shengfeng He. 2019. Single Image Reflection Removal Beyond Linearity. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019.* Computer Vision Foundation / IEEE, 3771–3779.

Xin-She Yang and Suash Deb. 2009. Cuckoo Search via Lévy Flights. In *World Congress on Nature & Biologically Inspired Computing, NaBIC 2009, 9-11 December 2009, Coimbatore, India.* IEEE, 210–214.

Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. 2018. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11207)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 418–434.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. 2017. Pyramid Scene Parsing Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017.* IEEE Computer Society, 6230–6239.

Bolei Zhou, Àgata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. 2014. Learning Deep Features for Scene Recognition using Places Database. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 487–495.