Water Simulation and Rendering from a Still Photograph Supplemental Material

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 DATASET FOR WATER SEGMENTATION

To train the water segmentation network, we first collect 1,995 real water images (1,895 for training and 100 for testing) to form a real water image dataset with ground truth annotation for water from multiple public scene datasets, SUN [Xiao et al. 2010], ADE20K [Zhou et al. 2017] and Places [Zhou et al. 2014]. However, such a small amount of training data results in overfitting and limited generalization capacity. Thus, we automatically augment the training data using a synthetic water image dataset with 10,000 randomly selected images from the texture image generation dataset described in Section B. To increase the diversity, we combine either background from the real water image dataset or foreground objects from the COCO dataset [Lin et al. 2014]. When using both datasets, we train the network for the same number of iterations on each of them. By combining the synthetic data with the real data, the segmentation accuracy is largely improved, which can be seen from the mean intersection over union (IOU) metric on the testing dataset, 0.8405 for the model trained with the synthetic data while 0.8160 for the model without such augmentation.

B DATASET FOR TEXTURE IMAGE GENERATION

We use the renderer described in Section 5 of the paper to create the dataset. The parameters are chosen randomly from a valid input range. The water surface color is chosen uniformly randomly from a valid subspace of HSV color space and then converted to RGB space. This limits the color range to be "blueish" while avoiding overly bright or overly vivid colors. The spherical harmonics constants are used to approximate the surrounding global lighting effects. We randomly generate a set of colors for several points on a unit hemisphere and use them to generate SH constants. Thus, we can generate infinitely many combinations of SH constants to ensure enough variety in lightning. In place of the reflection texture, we use an image that is flipped upside-down. Even though the reflection on water surface should be distorted in real scenes, the flipped image is a good approximation. We expect that what the reflection texture generation network should learn is irrelevant to such distortions. Based on this assumption, we chose images to be used as reflection textures from a subset of the Places database [Zhou et al. 2014] with outdoor scenes and sky images from the SWINySEG[Dev et al. 2019] dataset. In our dataset generation, we further made a simplification of the model. We assumed that the water region boundary is a straight line that lies within a random distance from the patch in image coordinates. This is required because each reflection image patch is supposed to be entirely in a water area and we do not need any information about the objects outside of the image patch.

Table 1: Parameter set optimized by the cuckoo search algorithm and used by our renderer.

Parameter	DoF	Range
Wind speed	1	[1.5, 30.0]
Wind direction	1	[0.0, 180.0]
Wave choppiness	1	[0.0, 3.0]
Camera height	1	[1.0, 75.0]
Camera angle	1	[45.0, 105.0]
Camera field of view	1	[45.0, 90.0]
Water color	3	[0.0, 1.0]
Level 0 SH lighting coefficients	3	[0.0, 2.0]
Level 1 SH lighting coefficients	9	[-1.0, 1.0]

C DETAILS OF TEXTURE IMAGE GENERATION NETWORK

As mentioned in the paper, the network architecture is based on UNet with residual blocks. The input patches are of size 224×224. The encoder consists of series of convolution layers with padding: 3×3 with stride 1, 7×7 with stride 2, 3×3 with stride 2, and 3×3 with stride 2; except the last convolution, each followed by batch normalization and a ReLU. There are five intermediate residual blocks which are followed by batch normalization and a ReLU. The decoder contains three upsampling layers of factor two. The first two upsampling layers are respectively followed by concatenation of feature map with the corresponding feature from the encoder, convolution, batch normalization, and a ReLU. The last upsampling layer is followed by feature concatenation and convolution layers. The convolution kernels in the decoder are 1×1, 3×3, and 1×1, in sequential order, all with stride 1.

D PARALLELIZED CUCKOO SEARCH

The energy computation is the most expensive component of our pipeline. It can be partitioned into frame rendering using the candidate parameter set and the DISTS and HSV color histogram energy evaluation. Rendering and DISTS evaluation accounts the majority of execution time. To improve efficiency, we perform evaluation for $k-1$th iteration and rendering for kth iteration concurrently (Algorithm 2). The concurrent execution of rendering and energy evaluation allows us to maximize the GPU utilization from 80-90% to 98%, and allows more energy evaluation per unit time. While the optimization process is not completely equivalent to the original serial version because of its lazy update feature, our experiments validate the effectiveness of the method (Figure 11). At the begging

ALGORITHM 1: Cuckoo Search

```
while termination condition not met do
    forall nest n_i do
        Get a cuckoo egg x'_i by Lévy flight from x_i.
        Choose a random nest n_j to lay the egg.
        if Score(x'_i) < Score(x'_j) then
            replace x_j with x'_i.
        end
    end
    forall nest n_i do
        Get a cuckoo egg x'_i by mutation from x_i
        if Score(x'_i) < Score(x_i) then
            replace x_i with x'_i.
        end
    end
    Replace the worst k eggs with randomly generated eggs.
end
return x_best
```

ALGORITHM 2: Parallel Cuckoo Search (kth iteration)

```
solutions_lévy_k ← generate_solutions_lévy(curr_nests)
solutions_mut_k ← generate_solutions_mut(curr_nests)
solutions_rand_k ← generate_solutions_rand(curr_nests)

frames_{k-1} ← receive_rendered_frames()
submit_rendering_tasks([solutions_lévy_k, solutions_mut_k,
    solutions_rand_k] )
[scores_lévy_{k-1}, scores_mut_{k-1}, scores_rand_{k-1}]
    ← evaluate_energy_function(frames_{k-1})
update_nests_lévy(solutions_lévy_{k-1}, scores_lévy_{k-1})
update_nests_mut(solutions_mut_{k-1}, scores_mut_{k-1})
update_nests_rand(solutions_rand_{k-1}, scores_rand_{k-1})
```

of evaluation, there is no significant difference of the energy curve due to the lazy update of candidate solutions. However, at later iterations, the difference in number of energy evaluations per unit time between the serial implementation and the parallel implementation lead to the difference of energy curves.

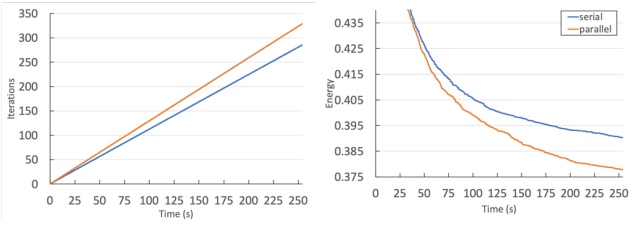

Figure 11: Comparison of the serial and parallel cuckoo search algorithms. All figures show average curves for 55 test images. On average, the termination condition is met after 269 iterations (212 seconds) with an average energy of 0.389.

(a) input **(b) w/o optimization** **(c) with optimization**

Figure 12: In most cases, such as in the top row, there is no sacrifice in quality when performing the reflection computation optimization. However, in some complex boundaries, such as in the bottom row, some artifacts can be noticed. The second input image is from Places [Zhou et al. 2014].

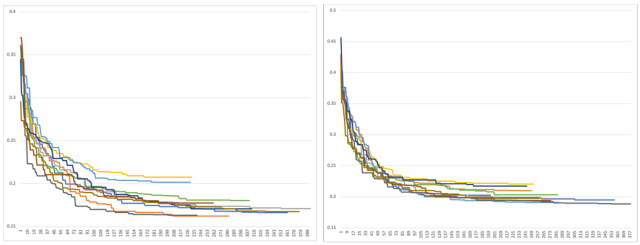

Figure 13: Optimization processes for the scenes in Fig. 1 and Fig. 8 first row with 10 different initial parameter sets, respectively. The vertical axis represents the energy, and the horizontal axis represents the number of iterations. Though the optimization process depends on the initial parameter set, they converge after enough iterations.

E INFLUENCE OF INITIAL PARAMETERS IN CUCKOO SEARCH

Because of the random nature of the search, the optimization process does depend on the initial candidate parameter set. However, we set a conservative enough threshold as the termination condition to let the optimization process reach small enough energy so that it will not give us a noticeable visual difference with respect to the initial candidate parameter set (see Fig. 13 and Fig. 14). We can choose an even stricter termination condition if desired.

F STUDY ON THE DESIGN OF THE ENERGY FUNCTION

In the design of the energy function, we chose the DISTS metric because it is not sensitive to local variations caused by wave dynamics, as opposed to other simpler image metrics. In Fig. 16, when the L1 loss is used as the energy function instead of DISTS, it fails

(a) input (b) best result (c) worst result

Figure 14: Comparison of the best and worst optimization results from Fig. 13. Our termination criterion is designed conservative enough and even the optimization processes that terminated with highest (worst) scores gives sets of parameters that give us plausible renderings. Input images: Edward Nicholl/Flickr (top) and - Paul -/Flickr (bottom).

to estimate the water dynamics completely, generating waves with either very high or very low frequency components only. These are expected results because we do not expect the wave dynamics of the input image and the rendered results to match exactly, and simple pixel-wise metrics cannot compute the perceptual difference between the two images. In addition to DISTS, the color similarity metric is employed to further penalize the color dissimilarity. In Fig. 16, we can observe that the addition of color dissimilarity energy lets the optimized parameters generate more visually plausible results by correcting the overall color shifts that are otherwise present in some results. We limit our discussion to the qualitative evaluation in this work because a quantitative evaluation would be difficult because of the local variations of water dynamics.

G RENDERING OPTIMIZATIONS

The basic method for computing the reflection color described in the paper is computationally inefficient mainly due to the ray marching step. We can reduce the number of iterations by using larger ray marching step sizes, but that can significantly degrade rendering quality. We observe that the collision point for the same reflection vector only changes when the viewpoint moves (user zooms or pans). Based on these observations, we can first precompute the ray marching step and reuse the results for multiple frames while the viewpoint is static.

Furthermore, in the precomputation step, we partition pixels in 4×4 groups in image space. For each group of 16 pixels, we precompute the potential collision points in screen space for 16 reflection directions uniformly distributed in the hemisphere, storing the results in a texture. In our implementation, we store at most two collision points (first and last) for each direction within each pixel group. The rationale is that the last collision should be against the same common "background" wall, while the first ray collision may be against different smaller walls. Finally, in the rendering pass, we retrieve the screen space collision points by interpolating the precomputed collision points in screen space instead of computing anew.

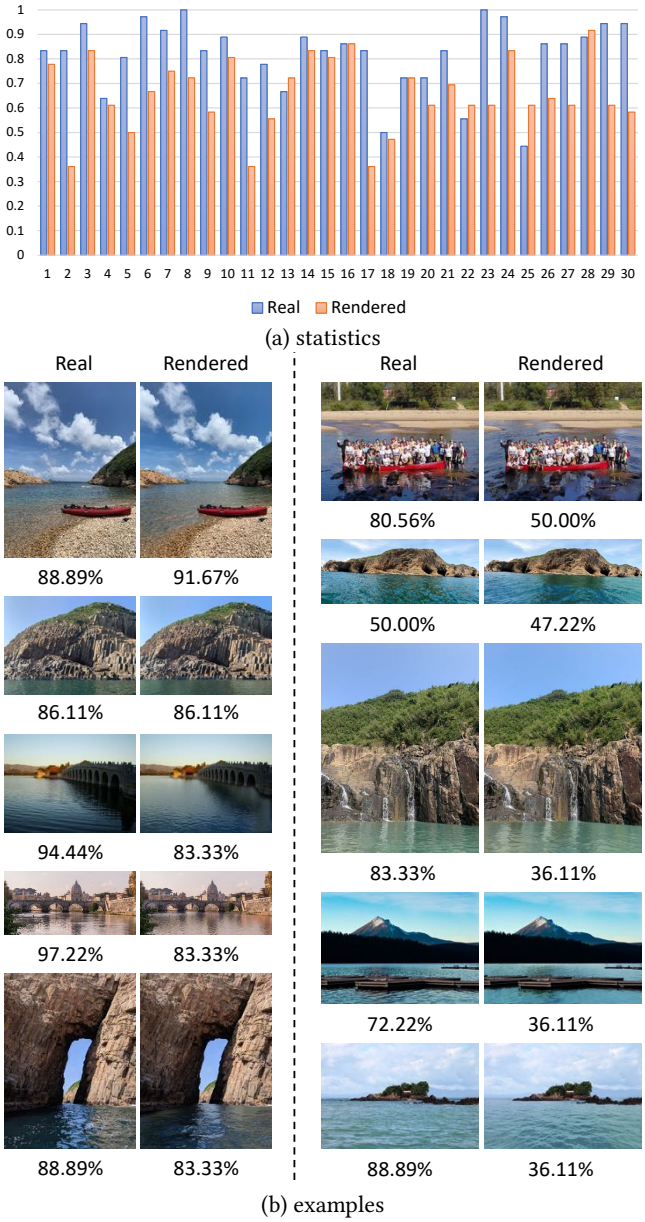

Figure 15: User study results. (a) illustrates the percentage of real/rendered images selected as real in every case; and (b) demonstrates the top 5 cases where the rendered images are thought as most photorealistic (left) and the top 5 cases where the rendered images are selected as least photorealistic (right). In (b), the numbers indicate the percantages of real/rendered images selected as real.

These optimizations increase the performance by approximately 10×, yielding accurate results in the vast majority of cases, but occasionally producing artifacts in complex boundary regions (Fig. 12). All results in the paper and supplemental material use these optimizations.

H DETAILS OF USER STUDY

The user study is designed to better evaluate our photorealistic synthesis results despite the difference in their water dynamics compared to the input images. So the test cases in the user study mix real images and synthesis results, which are then shown to users in random order. More specifically, we first randomly selected 30 input images from the testing dataset (which has 67 images in total) and then generated 30 test cases for every user. In each case, either the input image or its rendered result was shown to the user, who was asked to determined whether it was a photo or synthetic result. Users were given unlimited time to view the image and make a choice. A total of 72 users participated and 2160 answers were collected. The same number of real and rendered images were shown. The result shows that users successfully identified these real images as real 81.67% of the time, indicating that the users understood the task, while the rendered images fooled users in 65.46% of cases, which shows that most of our generated results are thought as realistic as real images. Fig. 15 (a) shows the percentage of real/render image selected as real in each case, and Fig. 15 (b) demonstrates five most photorealistic and five least photorealistic rendered images voted by users. Note that our method is able to generate photorealistic results even though the predicted water animation may not exactly match the input image.

REFERENCES

S. Dev, A. Nautiyal, Y. H. Lee, and S. Winkler. 2019. CloudSegNet: A Deep Network for Nychthemeron Cloud Image Segmentation. *IEEE Geoscience and Remote Sensing Letters* 16, 12 (2019), 1814–1818. https://doi.org/10.1109/LGRS.2019.2912140

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.

Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. 2010. SUN database: Large-scale scene recognition from abbey to zoo. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. IEEE Computer Society, 3485–3492.

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.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. 2017. Scene Parsing through ADE20K Dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 5122–5130.

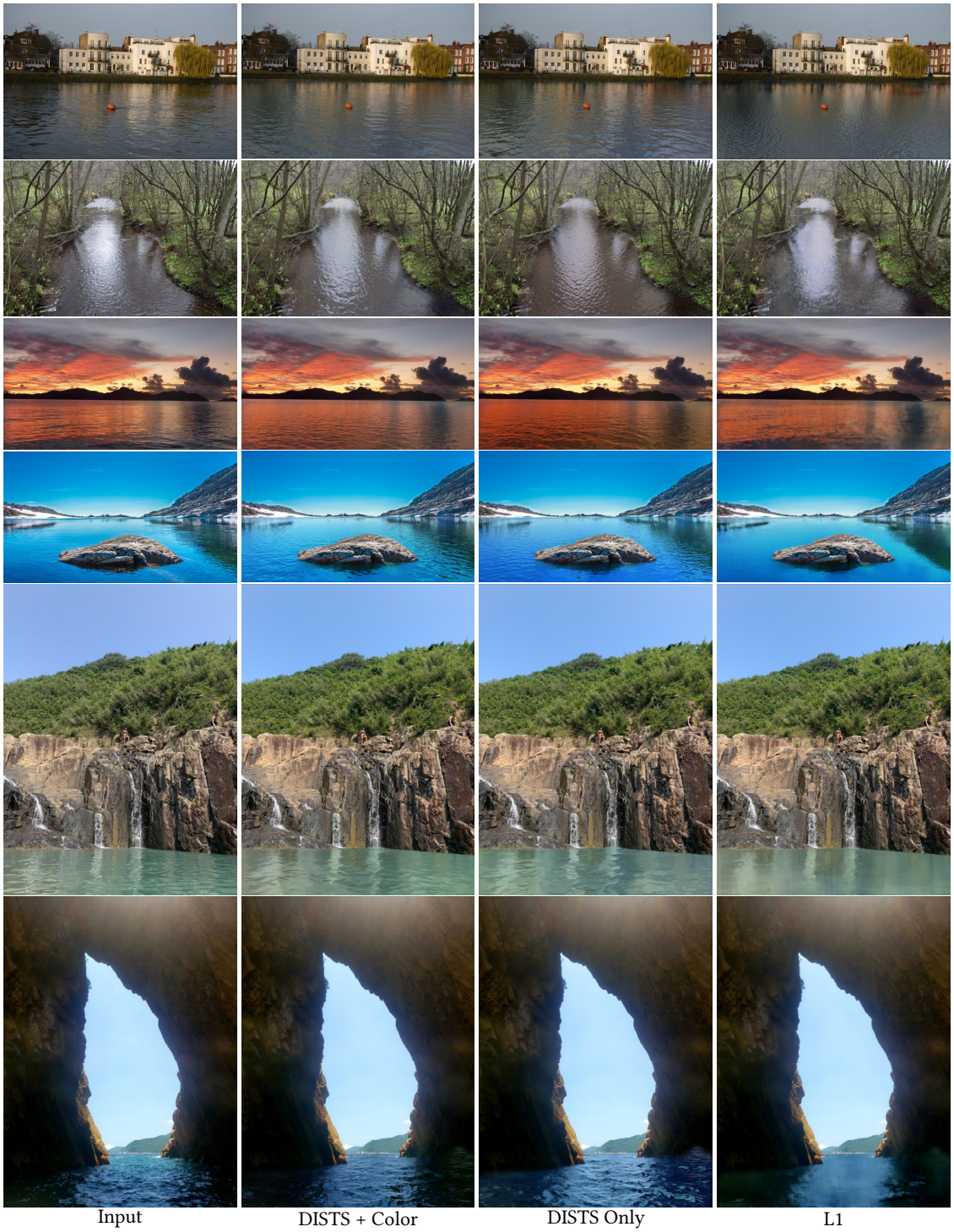

Figure 16: Comparison with parameters estimated with different energy functions for the cuckoo search. From left, inputs, results with the combination of DISTS and the color metric, results with DISTS metric only, and results with L1 loss. The results are obtained with 500 iterations for consistency. We obtain the best results using the combination of DISTS and the color metric. See the main paper for the water masks and the reflection textures used. The input images: - Paul -/Flickr (first), Edward Nicholl/Flickr (second), Foliez/Pixabay (third), and Unknown/PxHere (fourth).