OceaGAN: Realistic Texture Generation for Ocean Simulations with Generative Adversarial Networks

Yarkın D. Cetin, D. Yigit Polat
Department of Computer Engineering
Bilkent University
Ankara, Turkey
yarkin.cetin@bilkent.edu.tr yigit.polat@ug.bilkent.edu.tr

***Abstract*—We propose a novel approach for generating textures in ocean surface rendering applications. Appearance of surfaces like these are highly dependent on surface geometry and require tremendous amount of computation to be reconstructed visually in a digital medium with good accuracy. In our approach, we started with the assumption that a realistic ocean surface visual can be created by generating proper textures to be mapped on top of a naively formed wave mesh. Since analytical calculation of the texture from surface properties is not feasible, we chose a machine learning based approach where a generative model is used to generate these proper textures by using the surface geometry features as input. Deep neural networks have proven themselves in image generation tasks and can learn very complex patterns. Thus, we designed a deep generative convolutional neural network to dynamically generate textures for any given ocean mesh using its surface features. Our fully convolutional model can easily be integrated into real-time applications such as games and can be utilized in better looking ocean animations.**

I. Introduction

Ocean simulations are used extensively in entertainment and media, as the real oceans could not provide us with the freedom and comfort green screen studios can provide. However, for this freedom and comfort, the artists usually has to trade realistic for the pragmatic. Ocean simulations, with the advent of rendering, ray tracing and better simulations techniques such as IFFT based oceans [1] create realistic enough images for artists to use.

In this project we propose another such technique for creating realistic ocean renderings generated from simulated normal maps. We use a GAN [2] based approach which tries to approximate the ocean textures found in the nature.

In this project we draw inspiration from many previous papers on use of neural network architectures on generative models, such as images and art. Main inspirations come from works of Gatys et al., Nalbach et al. and Ledig et al.[3,4,5]

We build our generative model on ray traced representations of the simulated ocean. We believe that ray tracing is a good approximation to real image textures. Another thing we would like to add is while we exclusively use our model on Ocean textures, the model itself is actually domain free.

Our model utilizes a deep neural network capable of faster texture generation than conventional ray tracing methods. The model consists of three main parts. The generator network part which generates the texture given an input normal map. Second is the error network which uses a pre-trained network to back propagate better errors to our generator. Finally, we use a discriminator network which discriminates real-world ocean textures and ones generated by the generator model. This last network forces generator to create ocean textures which have similar texture statistics with the natural ocean textures.

Our model shows remarkable reconstruction ability of the original ray-traced texture. Also, with our full model which incorporates a GAN we were able to create textures which have more realistic foam and light interactions than the original one.

II. Related Work

There were some very interesting works on using neural networks for generating computer graphics. For smoke simulation, Thompson et al uses a convolutional neural network for speeding up the original computation times.

A more recent work uses “Deep Shading” [4] which shades the environment using color, specular and normal maps. This work demonstrates the ability of deep-learning based kernels to learn meaningful information from scene geometry and create textures for the objects. For texture generation without a particular geometry many other models are proposed, Ulyanov et al. [6] creates synthetic textures using a deep convolutional model. This model given multiple textures, can combine the textures in meaningful ways. Also, texture generation from normal maps can be seen as a super-resolution problem, where an image with higher amount of information needs to be created from an image with low amount of information. For this reason, we believe the work of Ledig et al. is important as it combines GANs with residual nets to super-resolve low resolution images. Our approach also adds MSE loss on top of the VGG loss used in [5]. Also our model differs in a way that it synthesizes real life images without any true image input.

III. Architecture and Method

A. Generator Network

As mentioned before the model draws inspiration from several models. We modify the architecture proposed in [5] to generate textures. The input of our network consists of 3-channel images representing the normal maps of the ocean mesh.

Output of the network is a 3-channel image which has the same size with the input normal map. Output images correspond to the colormaps of the ocean textures.

The network uses residual blocks proposed in [7] as building blocks. The skip connections in the residual network increases the gradient flow to the deeper layers and also reduces the amount of texture the model has to generate by using features from the initial normal map.
A residual network in our model consists of 2 convolution layers with a skip connection as seen in Figure 1.

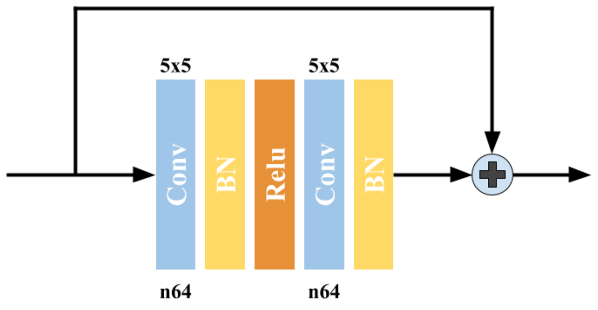

Fig. 1. An example for a residual block used in our model. Notice the lack of ReLU activation after the second convolution.

For non-linearity we use ReLU activation as it performs well on deep neural networks [8]. We also observed that downsampling the textures by a factor of 4 increases the effective area of the 5x5 kernels. This results in better texture reconstruction performance with ocean textures. Without the downsampling procedure, larger patterns in the ocean texture tends to washout because of the relatively small kernel size and comparatively shallower architecture compared to [5].
First, given a $\mathrm{H} \times \mathrm{W} \times 3$ image we project the 3-channel image into a $\mathrm{H} \times \mathrm{W} \times 16$ with convolution for better feature representation. For downsampling we use strided convolutions with kernel size of 3. With each convolution we double the feature space. With two consecutive strided convolutions we achieve two $\frac{\mathrm{H}}{2} \times \frac{\mathrm{W}}{2} \times 32$ and $\frac{\mathrm{H}}{4} \times \frac{\mathrm{W}}{4} \times 64$ sized images respectively.
We use 8 residual blocks for our model. We believe that it is a good trade-off between runtime speed and complexity of the model.
After residual blocks, the network is upsampled back to its original size using a symmetrical network. For upsampling we preferred using transpose convolutions instead of upsample-and-convolve approach prescribed in [9]. We did not encounter the possible artifacts generated by transpose convolution.

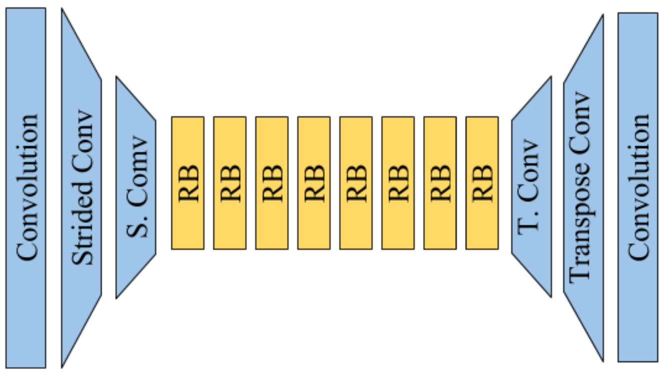

Fig. 2. The generator network with 8 residual blocks, the downsampler and upsampler networks.

For the output image we use the hyperbolic tangent function described in (2)

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2}$$

The tanh function ensures that all the values in the image are in the range of [-1,1] such that there is no oversaturation in the image. The generated image is then mapped to the [0, 255] range.

B. Loss Function

The training of our network needs a well behaving error function when reconstructing the images. In our model we use a combined loss function which has both a content and an realistic loss.

$$l_{combined}^{OT} = l_{content} + 5 \times 10^{-2} l_{realistic} \tag{3}$$

The loss network we describe here account for the content loss, i.e. how similar the generated texture to the real ocean texture perceptually.

The standard pixel-wise distance functions such as MSE and MAE did not provide enough information for the generator network to learn properly. The MAE and MSE is known for the mode collapse issues as shown in [9]. For this reason, we utilize the VGG similar to the loss mentioned in [5,9] creating our content loss network.

A pretrained VGG-19 model in [10] is used. The VGG model is trained on the ImageNet dataset and achieves a remarkable 7.3% top-5 error on ILSVRC-2012 dataset. The loss function is described in (3)

$$l_{VGG} = \frac{1}{W_{4,4} H_{4,4}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{4,4}(I^{baked})_{x,y} - \phi\left(G_\theta(I^{NM})\right)_{x,y} \right)^2$$

And the total loss $l_{content}$ is:

$$l_{content} = l_{VGG} + 5 \times 10^{-5} l_{MSE}$$

Where 4,4 denotes the 4th convolution in the 4th maxpooling layer of the VGG network and G_θ is our generator model. The

I^{NM} and I^{baked} denotes the normal map and the baked "real" ocean texture respectively. We also add the MSE error in the last term, we found that this improves color palette mapping from normal maps to ocean textures.

However, to input our generator images to VGG-19 which is specifically designed for 224x224 images, we rescale the output of our generator and the rendered ocean texture images to 224x224 using nearest neighbor interpolation. We found this rescaling does not disrupt the performance of our generator.

C. *Adversarial Loss Function*

The second term in our loss function is the adversarial loss which measures how "realistic network tries to discriminate between the real ocean textures and the ones generated by our generator network.

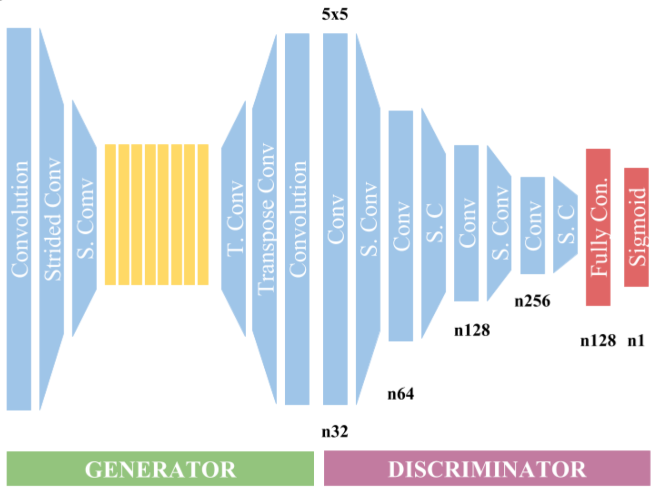

Fig. 3. The model with discriminator network and generator network. Note that the discriminator network does not use residual blocks.

The convolutions in the discriminator network use 3x3 kernels withy filter size doubling every two convolutions. The convolutional blocks provide both stride 2 and 1 convolutions making the network discriminate features in very large areas. The sigmoid output at the discriminator shows how "real" the image is, i.e. it shows how different it is than the real-life ocean pictures and textures fed into the network. The activations used in the discriminator are Leaky ReLU functions which have gradients for negative values as well while preserving their non-linearity.

Main advantages of the architecture is being fully convolutional. This makes it possible the train the network with rather small image patches (128x128 in our case) and without changing any parameter in the network reuse the same learned filter weights to generate textures for arbitrary sized images.

D. *Training and Hyperparameters*

For training the network we used previous guides on how to train GANs and convolutional neural networks in general. All implementation is done using Tensorflow. We selected Adam by Kingma et al. [14] as our gradient optimization algorithm. The parameters of Adam are $\beta_1 = 0.5$ and $\beta_2 = 0.999$. With a learning rate of 0.001 and a decay rate of 0.9 every 1000 iteration. For GAN training we first train the discriminator and then the generator with k=1, which means that the discriminator and the generator are trained equally. We trained our network for 4 hours using a mini-batch size of 16 and selected the model with the lowest training error.

The hardware we use for the experiments is a NVidia GTX1070 GPU mobile version. Since we use the VGG-19 model as well, a minimum of 6 GB memory is required which is satisfied by 1070. Our generation model however is smaller and can be run on GTX1060 as well.

IV. Datasets

In this part we describe our two datasets, the real ocean images which are collected from various resources manually and the ocean textures dataset, which consists of normal maps and the ray-traced colormap textures of an ocean simulation.

A. *Ocean Images Dataset*

This dataset is collected from the internet in Figure 4 you can see examples from the dataset. The dataset consists of 13 unique high-resolution ocean images. From these 13 images, we created 5000 128x128 by cropping the images randomly. Then the images are normalized such that the dataset has 0 mean and unit variance.

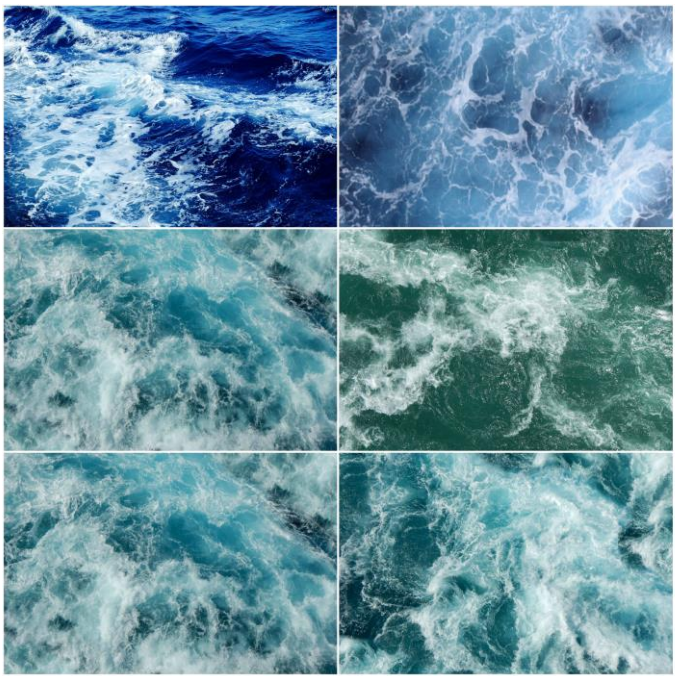

Fig. 4. 6 examples from the real ocean images dataset.

We believe this dataset captures the lighting and brightness changes around the foam structures well. Below you can see samples from the cropped images. Notice that there is a large variation.

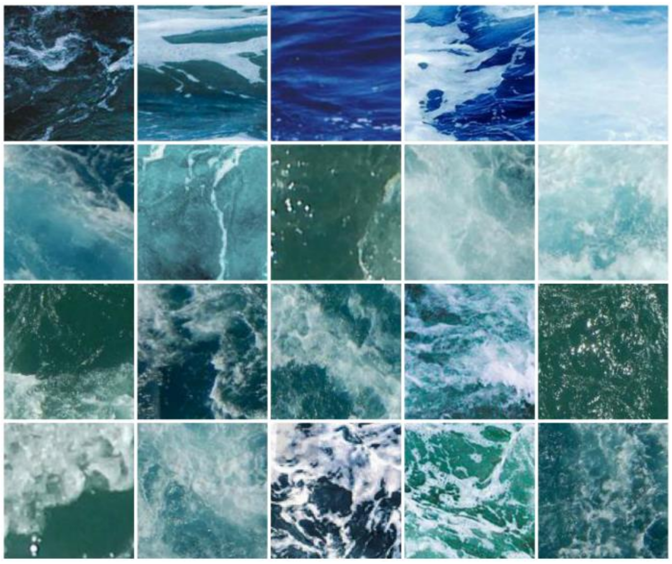

Fig. 5. 20 examples from the cropped real ocean images dataset. The images are collected manually, visually diverse images are selected for better performance.

B. Baked Ocean Texture and Normalmap Dataset

The baked textures for ocean textures are generated using the Blender Software. Blender is an open-source modeling, animating and rendering pipeline. Ocean Simulations in Blender are created according to a FFT based simulation prescribed by Tessendorf in [12]. A simulation corresponding to 25 second of ocean simulation is baked using the Blender built-in ocean simulator. From the simulated mesh, we rendered a 250 frames of ocean texture with ray-tracing. For the material we use our custom Ocean material consisting of Glossy, Refraction and Glass shaders. For better looking water, we add subsurface scattering and volumetric emission in the ocean such that the water looks realistic when bent and deformed with the waves.

The foam is generated from the FFT information and applied on top of the ocean surface as a texture. The foam model used in this Blender model does not have particle systems. Our model aims to regenerate these foams without the use of the direct FFT information.

In ray-tracing we use 128 samples per pixel with 1024x1024 images. We used Sobol distribution for the sampling as this creates more homogenous image quality than fully random sampling pattern [13].

For generating the dataset, we used 64 frames from the 250 frames generated with 4 frame intervals. The 4 frame intervals provided us with more temporal variance to train our model since the consecutive frames did not provide new information.

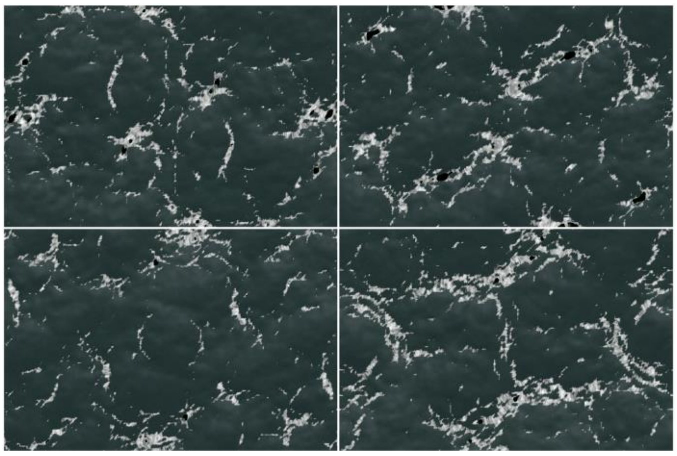

Fig. 6. 4 examples from the ocean textures dataset.

These images, similar to the ocean image dataset, then postprocessed to have 0 mean and unit variance. The images then cropped into 128x128 patches to increase the number of training examples to 5000.

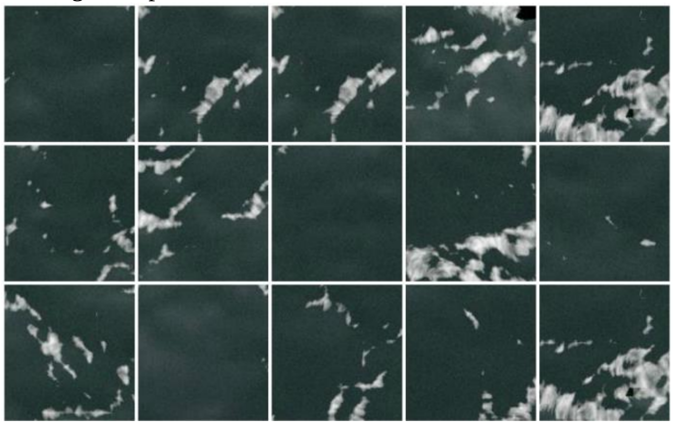

Fig. 7. 15 examples from cropped ocean texture dataset..

We also generated the normal maps using the Blender. The normal maps are also cropped and normalized like the colormaps textures.

A normal map is an image file where RGB color channels are used to encode corresponding XYZ values of the normal vector of the surface at that point. They are generally used for altering the lighting on a flat surface and making it look like it has fine surface features.

All X, Y and Z values are usually quantized in the range [0, 255] (except for the Z value in tangent space, it is in the range [128,255]) like RGB encoding but usually normalized to [0, 1] range in shaders for the sake of simplicity. While viewing a normal map as an image file via an image viewer, we usually see a dominant light blue color. The reason behind this is that flat regions in tangent space normal maps are encoded as X=128, Y=128, Z=255. When R, G, B is used instead of X, Y, Z respectively, it corresponds to that light blue color mentioned previously in RGB color space.

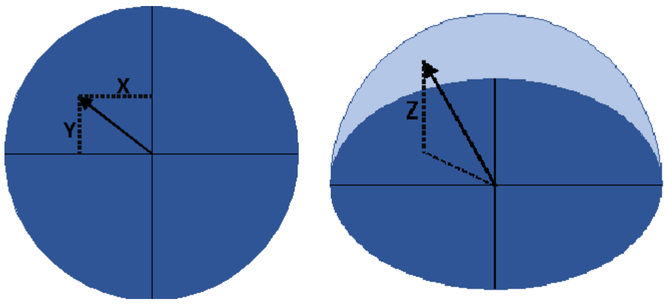

Fig. 8. An image showing the input of our model(left) the generated ocean textures (middle) and the original images captured from blender the model trying to reconstruct (right).

A normal map has much greater accuracy compared to a height map while defining fine surface details. To extract the gradients from the height map, we compute two different partial derivatives from the coarsely quantized height map and this causes great amount of detail loss. However, in a normal map, we have bidirectional surface derivatives that give much better encoding of small sized surface alterations assuming there are no irregularities and discontinuities in the height map.

Normal map values can easily be computed from a given surface mesh during vertex shading. Computed normal map can then be passed to the generative model to obtain a proper texture to be mapped on top of the ocean mesh.

$$Normal_X = \sin(\arctan(diff_X))$$
$$Normal_Y = \sin(\arctan(diff_Y))$$
$$Normal_Z = \sqrt{1.0 - (Normal_X)^2 - (Normal_Y)^2}$$

where $diff_X$ is the Z value difference between a mesh vertex and its horizontal neighbor and $diff_Y$ is the difference of the Z value of a mesh vertex and its vertical neighbor.

For test dataset we simulated a new 25 seconds of ocean with different random seeds but only generated the normal maps for this dataset. This dataset is uncropped as it will be fed into the trained network as complete frames.

V. Results

For results we did not combine the animation generation into one single pipeline where it can output directly from the blender. Instead for generating an animation we first generate the normal maps using the blender, then using these normal maps we generate textures using our model. Finally, the generated textures are imported into a real-time rendering software (Unity or OpenGL) to be played in real-time. While our complete pipeline does not provide real-time ray-trace quality texture generation. As we will demonstrate below, achieves much faster realistic ocean texture generation without performing ray-tracing.

A. Texture Generation with Training Data

First of all, we compare our results with images that have "ground truth" labels. The model actually tries to create more realistic images than the label images however a comparison on how good we can reconstruct the original content is important. In Figure 9, we observe that content is preserved as a whole however we some blurriness of the more detailed features can be observed as well.

One of the enhancements our model makes over the original rendered images is the lighting. Notice the lighting changes around the foam structures where the water becomes lighter color. This represent the thinning of the water layer around the wave "peaks" which network observed and learned to reproduce from real ocean images.

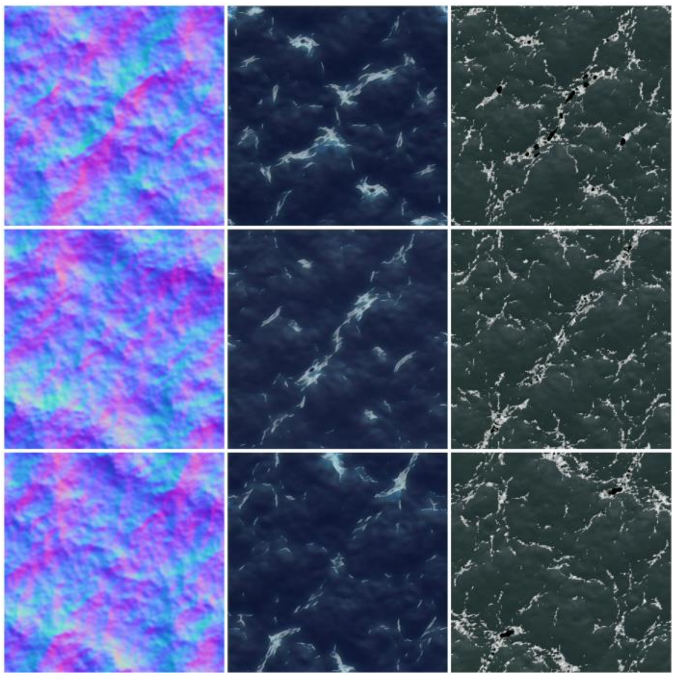

Fig. 9. An image showing the input of our model(left) the generated ocean textures (middle) and the original images captured from blender the model trying to reconstruct (right).

B. Texture Generation with Novel Data

We give novel normal maps to our network and observe its outputs. As we can see the network generates images similar to those generated by the blender renders.

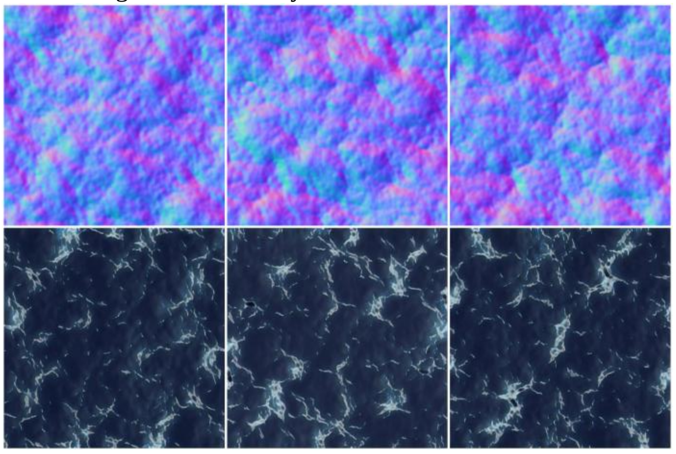

Fig. 10. An image showing output of our model (bottom) with novel normal maps (bottom). The similar performance to the training set can be observed.

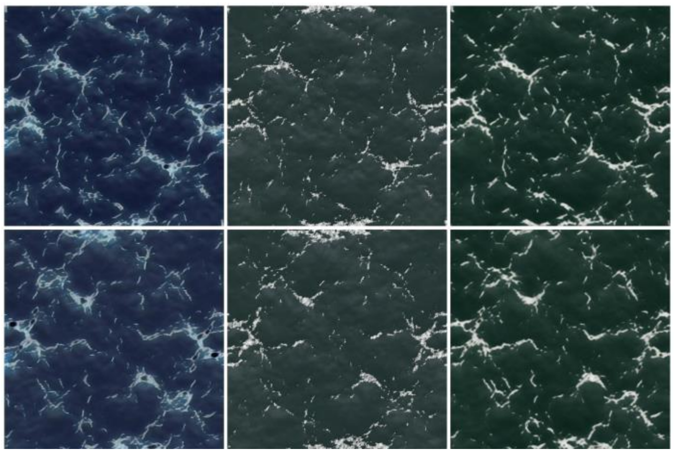

Fig. 11. An image showing effects of using GAN loss, the output generated by our network which uses GAN loss(left) the ground truth (middle) and the output generated by another version of our network which does not use GAN loss in training (left). We show that given a normal map of the ocean surface, we can create almost perfect reconstructions in much less time.

C. *Controlling the Foam Generation*

A disadvantage of our model is its rigidity. When we train a model on certain settings of ocean material and foam amount, the network only learns to generate the images at these particular settings. Changing settings in conventional renderers does not consume time, but we to change settings in neural networks might mean a complete retraining of the model, costing hours of computation time.

Our network while cannot control the hue or saturation, can control the amount of foam generated in a given ocean geometry. By controlling the mean value of the normal map input, we can control the amount of foam generated into the texture easily. Figure 12 shows different amount of foam generated by tuning the normal map, using the same model parameters.

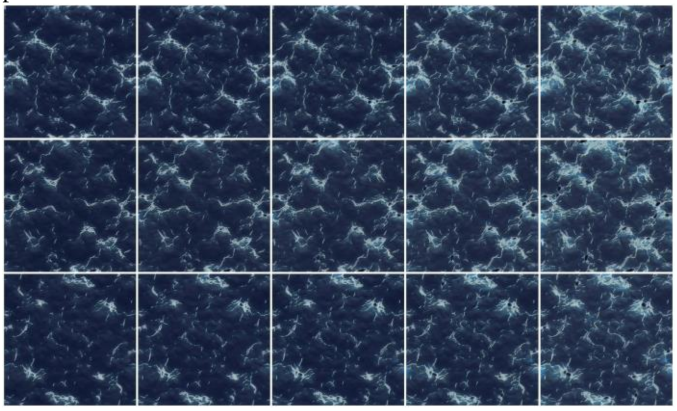

Fig. 12. An image showing the change of foam amount, adding a mean-shift of 0.03 to normal map each time changes the behavior of the model. Notice the artifacts occuring at higher mean shifts, which are caused by oversaturation.

D. *Time Performance Analysis*

Our main achievement is making the texture generation faster by utilizing the power of GPU computed neural networks. While we could not manage to combine the separate parts into one single pipeline where we can input the ocean geometry and produce textures, we achieved faster texture baking than the conventional ray-tracing method.

TABLE I. PERFORMANCE OF THE MODEL

Resolution	256x256	512x512	1024x1024
Ray-tracing(ms)	~470ms	~1130ms	~5400ms
Ray-tracing(fps)	~2.12fps	~0.88fps	~0.18fps
OceaGAN(ms)	~12ms	~35ms	~110ms
OceaGAN(fps)	~83 fps	~28fps	~9fps

In Table I. we deliberately left out the overhead coming from the simulation of the ocean as these simulations can be run in real-time as demonstrated in [15]. The results show that our model increases the speed of texture baking by 40 to 50-fold. This is a great improvement over the naïve ray-tracing approach.

VI. CONCLUSION AND FUTURE WORK

Generating high quality textures for surfaces with complex refractive capabilities is an important yet difficult undertaking in computer graphics. In or project we tried to approach this problem with power of generative neural models. Our model exploits the regularities in how ocean water behaves under ambient light and how it generates foams under specific geometry. Combining residual and generative adversarial networks, we demonstrated that we can generate realistic ocean textures only using the ocean geometry, i.e. normal maps of the surface. Using our method, we achieved 40-fold faster texture generation than ray-tracing. While our model lacks end-to-end simulate and generate pipeline, we believe these can be reduced to engineering problems which we can solve, given time. There are further improvements to our model, which we will briefly discuss.

First of all, the current model does not support specular reflections and instead counts on the texture shaders to handle such reflections. Our model could be improved such that it can also compute specular reflections, given the positions and locations of the light sources in the scene. However, training of such a model would be very costly in terms of computational resources and it is the main reason we were unable to deliver at this time.

The second improvement for the model is the creation of particle-based foam and water. As it in its current form our model only supports texture-based foam. An addition to our network which outputs an image mask with vectors for each pixel could dictate the rendering engine to emit particles.

VII. References

[1] S. Hochrieter, and J. Schmidhuber. Amplitude Malformation in the IFFT Ocean Wave Rendering under the Influence of the Fourier Coefficient. Electronics, Vol. 18, No. 2, December 2014

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville Yoshua Bengio. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]

[3] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Mingli Song. Neural Style Transfer: A Review. arXiv:1705.04058 [cs.CV]

[4] Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, Tobias Ritschel. Deep Shading: Convolutional Neural Networks for Screen-Space Shading. arXiv:1603.06078 [cs.GR]

[5] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. arXiv:1609.04802 [cs.CV]

[6] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, Victor Lempitsky. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. arXiv:1603.03417 [cs.CV]

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[8] Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network arXiv:1505.00853 [cs.LG]

[9] Agustinus Kristiadi, Why does L2 Reconstruction Loss Yield Blurry Images?. https://wiseodd.github.io/techblog/2017/02/09/why-l2-blurry

[10] Very Deep Convolutional Networks for Large-Scale Image Recognition K. Simonyan, A. Zisserman arXiv:1409.1556

[11] Goodfellow, Ian J. et al. “Generative Adversarial Networks.” CoRR abs/1406.2661 (2014): n. pag.

[12] Tessendorf, J. 2001. Simulating ocean water. In Simulating Nature: Realistic and Interactive Techniques. SIGGRAPH 2001 Course Notes 47.

[13] Niederreiter, Harald. "Quasi-Monte Carlo methods and pseudo-random numbers." Bulletin of the American Mathematical Society 84.6 (1978): 957-1041

[14] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[15] Wang, Chin-Chih, et al. "Ocean wave simulation in real-time using GPU." Computer Symposium (ICS), 2010 International. IEEE, 2010.