# Mug Makeover:
# Generating Realistic Ray-Traced Images from Rasterized Renders

Thomas Yim*
Stanford University
yimt@stanford.edu

Daniel Lee*
Stanford University
jaeylee@stanford.edu

Caroline Cahilly*
Stanford University
ccahilly@stanford.edu

## Abstract

*Ray tracing, a relatively novel technique in image rendering, surpasses traditional raster renderers by creating remarkably realistic scenes. However, due to its intricacy, it inherently suffers from significantly slower processing speeds compared to raster renderers. In this research, we propose a novel approach that leverages machine learning and computer vision to bridge the gap between rasterized and ray-traced images. Our methodology focuses on predicting a ray-traced version of a render from a rasterized input using a conditional generative adversarial network (cGAN) derivative, specifically pix2pix.*

*To facilitate our study, we curate a dataset of mug renders, forming the foundation for training and evaluation. We explore five variations of our model and extensively analyze their performance. Notably, the best-performing model incorporates depth and normal maps in addition to the rasterized image during the generation process. By utilizing our model, we achieve a remarkable enhancement in efficiency, generating ray-traced-like outputs from rasterized inputs approximately 4.5 times faster than the conventional ray tracing method while resulting in significant quality improvements from rasterized images. We assess the fidelity of our model by subjecting the outputs to rigorous evaluation metrics, including passing them through a pre-trained convolutional neural network (CNN)-based discriminator model and analyzing statistical measures alongside qualitative assessments.*

## 1. Introduction

We aim to leverage deep learning methods for computer vision to generate ray-traced images from rasterized images. Ray tracing is a rendering technique used in computer graphics that can realistically simulate the lighting in a scene. It generates 2D images from 3D models by tracing the path of a large number of light rays from the view camera, through the 2D viewing plane, out into the 3D scene, and back to the light sources while accounting for reflections, refractions, shadows, and indirect lighting [7]. Thus, the process is very computationally expensive.

By contrast, rasterization is a much faster rendering technique than ray tracing that is commonly used in real-time computer graphics. It similarly converts 3D models, represented as a collection of polygons, into 2D images by determining the visibility and color of individual pixels through a process of scanning and sampling. However, the images generated appear much less realistic than those generated by ray tracing methods because it struggles with features like shadows and reflections [7].

Thus, we wanted to see if we could train neural networks to quickly generate ray-traced images from rasterized images, in turn speeding up the entire ray tracing process. We hypothesized that neural networks could learn an approximation of the ray tracing algorithm that looks relatively indistinguishable to the human eye but avoids the complex process of tracing a large number of light rays.

Our final goal is to train a neural network to generate ray-traced images based on rasterized images. First, we will train a convolutional neural network (CNN) to classify images as either ray-traced or rasterized. We do so to see if a CNN can learn the difference between the two modes and use it as a discriminator in the future. Then, we will train a pix2pix-based model, which is a type of conditional generative adversarial network (cGAN), to generate ray-traced images based on rasterized images. Therefore, it takes a rasterized image as input and predicts a ray-traced image as output. We will experiment with a variety of inputs and model structures to find the most optimal way to generate the ray tracing approximation output.

## 2. Related Work

### 2.1. Approaches to Approximating Ray Tracing

As ray tracing is a computationally expensive task, there have been past attempts at approximating its effects. None of the following common approaches use machine learning.

*Equal contribution

- **Screen Space Reflection (SSR)**: This technique uses data that the rasterization process has already generated, such as the depth map, to approximate the reflection of objects in the scene [2]. There are three steps in SSR: ray casting, reflection sampling, and blending. Ray casting casts a ray in a specific direction starting from each pixel on the screen to determine the reflection ray. Reflection sampling samples the information about the reflected color as the reflection ray intersects with the scene. This sampled color is blended with the original pixel color based on factors like surface roughness and material properties. While SSR provides a good approximation of reflections, it does not capture distant reflections well and does not have any non-reflection features of real ray tracing.

- **Screen Space Ambient Occlusion (SSAO)**: This technique approximates ambient occlusion (the soft shadowing effect caused by indirect lighting) by first sampling nearby pixels for each pixel on the screen, sampling depth values of those sampled pixels and then calculating the occlusion factor based on that information to either darken or lighten up the pixel [1]. While SSAO is helpful in approximating ambient occlusion information, it fails to incorporate other ray tracing features such as realistic reflections and depth of field.

- **Screen Space Global Illumination (SSGI)**: This technique only traces rays from each pixel on the screen to nearby surfaces instead of tracing every global light ray. It then calculates the color of each pixel based on the light that is reflected off of those surfaces. Because this process is repeated for each frame, it allows for the lighting to change dynamically as the scene is animated, which is why it is commonly used in real-time rendering [8]. Even though SSGI can significantly improve the realism of real-time rendered scenes by adding indirect lighting, it can produce artifacts like halos around light sources and is not as accurate as ray tracing.

We were not satisfied with any of these techniques as they were only able to approximate just one feature of ray tracing instead of approximating every feature.

### 2.2. Lighting Correction and Illumination

Machine learning methods have been used to approach lighting correction and illumination problems. These approaches are relevant to our project since our project can be viewed as illuminating and correcting the lighting on rasterized images in order to approximate ray-traced images. As one example of lighting correction, Chen *et al*. use end-to-end trained fully convolutional networks to generate bright long-exposure images from dark short-exposure images in their paper [3]. They found that CNNs performed better than traditional lighting correction pipelines, which use a combination of strategies like denoising, sharpening, and demosaicing. Many others have used CNN-based models to enhance images in low-light settings, such as Guo *et al*. and Li *et al*. [5, 9].

Similar attempts have been made to illuminate a variety of settings, such as the work done by Gardner *et al*. [4]. They aim to estimate the illumination information of an image using a deep CNN. Then, they insert a 3D object into the scene. They feed the new image and the lighting information of the original image (outputted by the CNN) into a rendering engine in order to relight the inserted objects. Here, they demonstrate that CNNs can learn lighting information useful in rendering.

### 2.3. Large-Scale Ray-Traced Datasets for NeRF

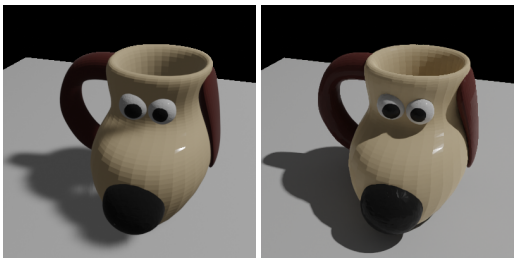
Datasets mapping rasterized images to ray-traced images did not appear in any popular literature. So, we instead examined work within the field of deep learning that uses large-scale ray-traced datasets. The most notable example was created by Tremblay *et al*. [11]. They use a synthetic dataset of about 300,000 1600x1600 images across 2,000 complex scenes, which is orders of magnitude larger than previous datasets. With this dataset, they train a NeRF (neural radiance fields) model to perform novel view synthesis, meaning that they use images from a scene to generate images of that scene from new viewpoints. The generated novel views notably contain realistic lighting, suggesting that the NeRF model was able to pick up on the lighting features and how they changed from view to view.

### 2.4. Image-to-Image Translation Models

A type of cGAN called "pix2pix" aims to solve a similar problem to ours. Specifically, Isola *et al*. modify a generative adversarial network (GAN) model such that it conditions on an input image when generating an output image [6]. So, it is trained on input-output image pairs. Interestingly, Isola *et al*. use dropout at test time to insert noise into the model so that the output at test time is always randomized to be slightly different.

Other approaches to image-to-image translation exist, such as work done by Zhu *et al*. [13]. Their objective is similar to pix2pix, except that they do not assume that the training data consists of paired input-output images. Rather, they only assume that the inputs consist of a set of images from one class and the outputs consist of a set of images from another class.

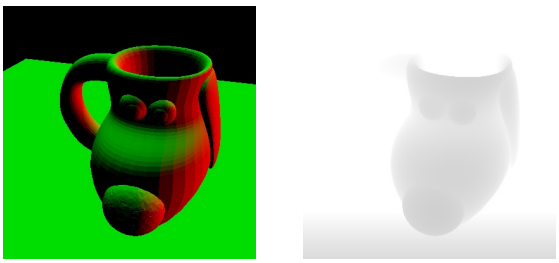
## 3. Dataset and Features



(a) Rasterized (b) Ray-Traced

Figure 1. Example renderings

Our dataset contains 117 mugs, with a train-val-test split of 82-18-17. We use a wide variety of mug types, from very traditional-looking mugs to more unique mugs, as shown in Figure 1. Each mug has 200 images rendered from 100 different evenly spaced viewpoints around the mug, with one rasterized and one ray-traced image per viewpoint. Thus, we have 23,400 total images between the two classes. Each image is 256x256 pixels.



(a) Normal Map (b) Depth Map

Figure 2. An example normal and depth map

Later on, we also acquired the normal and depth map for each viewpoint of each mug (Fig. 2), bringing the total image count to 46,800. Normal maps provide information about the texture of the object/scene at a specific viewpoint by encoding the direction of the surface normals in two-dimensional RGB images. Depth maps are grayscale images that represent the depth information of the object/scene from a specific viewpoint.

We based our method on generating the dataset from the method used by Tremblay *et al*. [11]. Specifically, we generated our data set synthetically in Blender using 3D object files downloaded from Objaverse, a large 3D object database hosted by the Allen Institute for AI [10]. We used the Eevee rendering engine to generate the rasterized images and the Cycles rendering engine to generate the ray-traced images.

The average rendering time per image for Eevee was 0.21 seconds, which is significantly faster than the rendering time per image for Cycles, which was 1.68 seconds. It took 0.04 seconds on average to generate a depth map and 0.04 seconds to generate a normal map.

## 4. Methods

### 4.1. CNN Classifier Model

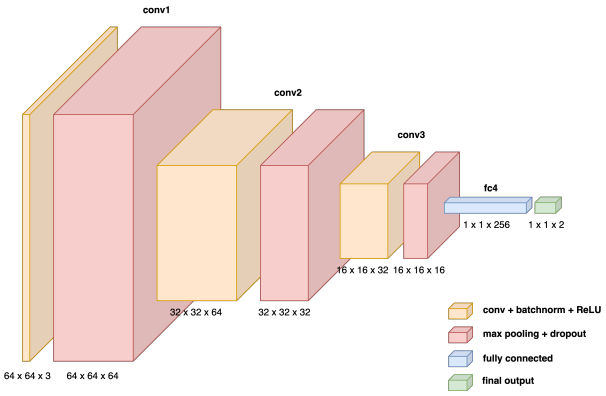



Figure 3. Our CNN architecture

We trained a CNN as shown in Figure 3 to classify images as ray-traced or rasterized. For conv1, our filters were 5x5 with a padding of 2, and for conv2 and conv3, our filters were 3x3 with a padding of 1.

$$L = -\sum_{i=1}^{n} \log\left(\frac{e^{f_{y_i}}}{\sum_{j=1}^{n} e^{f_j}}\right) \tag{1}$$

For our loss function, we used the cross-entropy loss as shown in Equation (1). We use cross-entropy loss because we want to maximize the normalized probabilities that each image in the batch is classified correctly. To optimize this loss function, we use the Adam optimizer in order to take advantage of per-parameter learning rates and momentum.

With this relatively simple model, we expected that we could achieve above 90% test accuracy because related works discussed in Section 2, such as NeRF, seemed to learn good representations of the lighting features of objects [11].

To evaluate our classifier, the most important quantitative metric was test accuracy because we wanted an algorithm that performed well on unseen examples of ray-traced and rasterized images. We also generate and examine saliency maps to see which areas of the image the CNN was focusing on for classification.

### 4.2. pix2pix Model

We base our model on pix2pix, the cGAN discussed in Section 2. Thus, we input rasterized images into pix2pix, which the model then conditions on to generate ray-traced images. The model structure has two parts, the generator

and the discriminator, which are trained concurrently as described later on. The generator is a U-Net where the encoding layers downsize the input image and the decoding layers upsize the image back to its original shape to produce the output. Skip connections are used to help preserve some of the fine-grained details of the image from original to output. The discriminator is a PatchGAN discriminator that only looks at local patches for information. It takes as input the channel-wise concatenation of the input to the generator and either the output of the generator or the corresponding ground-truth image.

The train-val-test split is the same as that used for the CNN classifier.

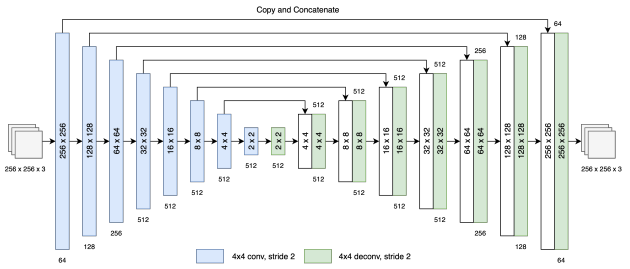



Figure 4. pix2pix Base architecture

In our experiments, we use five variations of the pix2pix model. We based our pix2pix model on Jun-Yan Zhu and Taesung Park's implementation [6]. Unlike the original pix2pix model, we wanted deterministic output rather than randomized outputs. We accordingly modified the model to remove dropout layers at test-time [6]. We also wrote custom data loaders for our modified inputs and edited the model code to experiment with the various model structures. Lastly, we edited the training logic to experiment with various hyperparameters. Below are the five models we used:

1. **Base Model**: For our base generator architecture, we use the U-Net architecture depicted in Figure 4. Here, the input was just the rasterized images (256, 256, 3), and the output was the predicted ray-traced images (256, 256, 3).

2. **Depth Model**: For this model, we modified the base model to take as input the channel-wise concatenation of the rasterized image and the depth map, giving an input shape of (256, 256, 4). The output is still (256, 256, 3).

3. **Depth Model with Added Skip Connection**: For this model, we add one final skip connection to the depth model. Specifically, we add a skip connection from only the rasterized image (the first three channels of the input) to the output from the depth model. We do this to ensure that the rasterized image matters more than the depth map when generating the output. Then, we do a 1x1 conv layer with three filters to get our final output of shape (256, 256, 3).

4. **Depth + Normal Model**: For this model, we modified the base model to take as input the channel-wise concatenation of the rasterized image, the depth map, and the normal map. So, the input shape is (256, 256, 7). The output is still (256, 256, 3).

5. **Deeper Depth Model**: This model is a deeper version of the depth model. Specifically, after each encoder layer from the depth model, we added a conv layer with the same number of filters as the previous layer. However, we use 3x3 filters with padding of 1 and stride of 1 to preserve the spatial dimension. Likewise, after each decoder layer, we add a deconv layer with the same hyperparameters. We did not add skip connections between these new layers. When evaluating the depth model on our test and train images, it only performed marginally better on the train images. Thus, we wanted to see if making our model deeper would give the model more expressibility to learn the lighting features.

The objective of our model is to generate fake ray-traced images that look as real as possible based on the input rasterized image. Thus, the loss function for our U-Net works in two steps.

$$L_D = -\mathbb{E}_{x,y}\left[\frac{\log\sigma(D(x,y)) + \log(1-\sigma(D(x,G(x))))}{2}\right] \tag{2}$$

First, we calculate the gradients for the discriminator and update its weights. The discriminator wants to maximize the likelihood of guessing that true ray-traced images are true and false ray-traced images are false. So, to calculate its gradients, we use binary cross entropy loss with logits on the predictions of the generator for the true ray-traced images and the false ray-traced images as shown in Equation (2).

$$L_G = L_{L1} + L_{\text{cGAN}} \tag{3}$$

$$L_{L1} = \lambda \cdot \mathbb{E}_{x,y}\left[||y - G(x)||_1\right] \tag{4}$$

$$L_{\text{cGAN}} = -\mathbb{E}_x\left[\log\sigma(D(x,G(x))\right] \tag{5}$$

Second, we calculate the gradients for the generator and update its weights. The generator wants to maximize a combination of two things: (1) the similarity between the generated images and the true ray-traced images and (2) its ability to trick the discriminator into thinking that fake images are real. The loss function which allows us to achieve this objective is shown in Equation (3). To measure similarity, we

use L1 loss as shown in Equation (4), which is the mean of the average pixel-wise absolute difference of all of the fake-real pairs. To measure the ability of the generator to trick the discriminator, also known as the adversarial loss, we again use binary cross entropy with logits as shown in Equation (5). However, this time we have the input as the false images only, and the target is that it is real.

For our optimizer, we again use Adam to take advantage of per-parameter learning rates and momentum.

We expected the model to pick up on key lighting and shadow features of ray-traced images. For example, the ray-traced images have more defined shadows with sharper edges compared to rasterized versions, as demonstrated in Figure 1. However, we expected certain flaws in our generated images, such as improperly-shaped shadows or GAN artifacts like blurriness in at least the first few versions of the model, as the basic pix2pix model would need tuning to learn and improve on this lighting task.

### 4.3. pix2pix Evaluation Metrics

To evaluate this model on unseen examples, we considered the following metrics computed for the test set:

- **Mean L1 per image**: Our pix2pix model uses L1 loss to do a pixel-wise comparison of the generated image and the true ray-traced image during training and validation. We compute the same metric for the test set, considering the average L1 loss for each image, to see how the model performs on unseen examples. The range for mean L1 distance is [0, 255], with 0 being the best.

- **Accuracy of the CNN classifier we trained**: Since our discriminator is not trained simultaneously like the discriminator used in pix2pix, it allows us to test the effectiveness of our generative model in "tricking" the discriminator to predict that our fake image is ray-traced instead of rasterized. The range for the CNN accuracy percentage is [0, 100], with 100 being the best.

- **Mean Structural Similarity Index (MSSIM)**: This metric gives a quantitative measure of visible differences between a distorted image and the true image. It attends to differences in the luminance of the pixels, the contrast between pixels, and the overall structure, which are all features that heavily influence human perception. It works best when applied locally for numerous reasons, including that people can only perceive small areas of images at high resolution from typical viewing distances. Within each local region, higher weights are assigned to the pixels closer to the center, and the weights gradually decrease for pixels farther away. The weightings are based on a Gaussian distribution, and $\sigma$ controls its spread. We used $\sigma = 1.5$, which empirical results suggest is most effective [12]. The final output of MSSIM is the mean of all the locally-calculated SSIM values. Empirical results suggest that locally calculating SSIM values by using an 11x11 window with stride 1 often works best, which is what we used [12]. The range is [-1, 1], with 1 being the best.

- **Qualitative observation of outputs**: We examined generated test examples to see if there are certain failures of the model visible to the human eye, such as improperly-shaped shadows, strange lighting, or blurry artifacts.

- **Saliency Map Analysis**: Using the PatchGAN discriminator's binary cross-entropy loss, we generated a saliency map of the input image to see which pixels had the greatest impact on the loss. To generate the saliency map, we get the loss, then backpropagate to the input and find the absolute value of the gradient of just the fake-generated image's channels. We then take the max value at each pixel across these three channels to create the saliency map.

- **Inference time**: We compared the inference time of our model plus the time Eevee takes to generate an example (and associated depth and normal maps) to the time Cycles takes to generate an example. We do so to test if we can generate approximate ray-traced images from 3D models more quickly than the Cycles rendering.

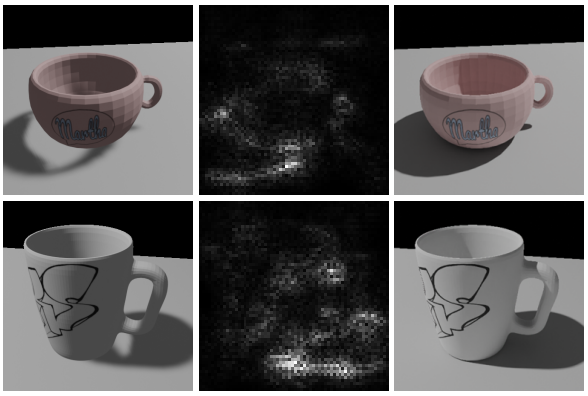
## 5. Experiments/Results/Discussion

### 5.1. CNN Classifier Results

For our loss function, we used cross-entropy loss, and for our optimizer, we used Adam with the learning rate set to 0.001. We set the dropout rate to 0.1. Before training and test time, we first resized the images to 64x64 pixels. We trained the model for 10 epochs. We obtained these hyperparameters through tuning on the validation set.

The CNN was able to achieve a 93.97% accuracy on the test set, suggesting that our model generalized well to unseen examples. We determined this was satisfactory for our use. We now have a way to measure our main model's performance: the percent of all generated images classified as ray-traced by the CNN will be the approximate performance, as it means the CNN (which is supposed to mimic a human in differentiating between rasterized and ray-traced images) was fooled into thinking that the image was ray-traced. This CNN will test if an image passes as ray-traced instead of rasterized, whereas the cGAN's discriminator will attempt to distinguish between a real and generated image.

#### 5.1.1 CNN Saliency Map



(a) Eevee rendering (b) CNN saliency map (c) Cycles rendering

Figure 5. An example saliency map compared to its respective Eevee and Cycles render

The saliency generated for the CNN classifier helps us examine the different areas the CNN is focusing on in order to classify the images.

As seen in Figure 5, the CNN classifier is focusing on the shadows, as an imperfect shadow with holes and a smooth shadow is the biggest difference between rasterized and ray-traced images. We further observe that the CNN also focuses on the edges of the mug as well as its handle, as those areas are likely to be affected by the mug's own shadow.

### 5.2. pix2pix Results

#### 5.2.1 Quantitative Observation of pix2pix Results

| Model | Mean L1 | CNN (%) | MSSIM |
|---|---|---|---|
| Ground Truth | 0 | 100.00 | 1.000 |
| Base | 4.168 | 99.88 | 0.925 |
| Depth | 4.026 | 99.18 | 0.928 |
| Depth+Skip Connection | 4.823 | 100.00 | 0.896 |
| Depth+Normal | 3.730 | 99.65 | 0.934 |
| Deeper Depth | 5.770 | 100.00 | 0.919 |

Table 1. Quantitative statistics of pix2pix results

As seen in Table 1, the Depth + Normal model performed the best, achieving the lowest mean L1 loss and highest MSSIM score among all of our models. While the CNN accuracy was a bit lower than some of the other models, we believe that this is within the margin of error, as our CNN was not perfect to start with. Assuming that the CNN accuracies are all within the margin of error, the second best model was the Depth model, followed by the base model. The Deeper Depth model and the Depth + Skip Connection model performed poorly, indicating that changes to the model architecture were not helping the performance.

#### 5.2.2 Qualitative Observation of pix2pix Results

Figure 6 shows a comparison of models across three example mugs.

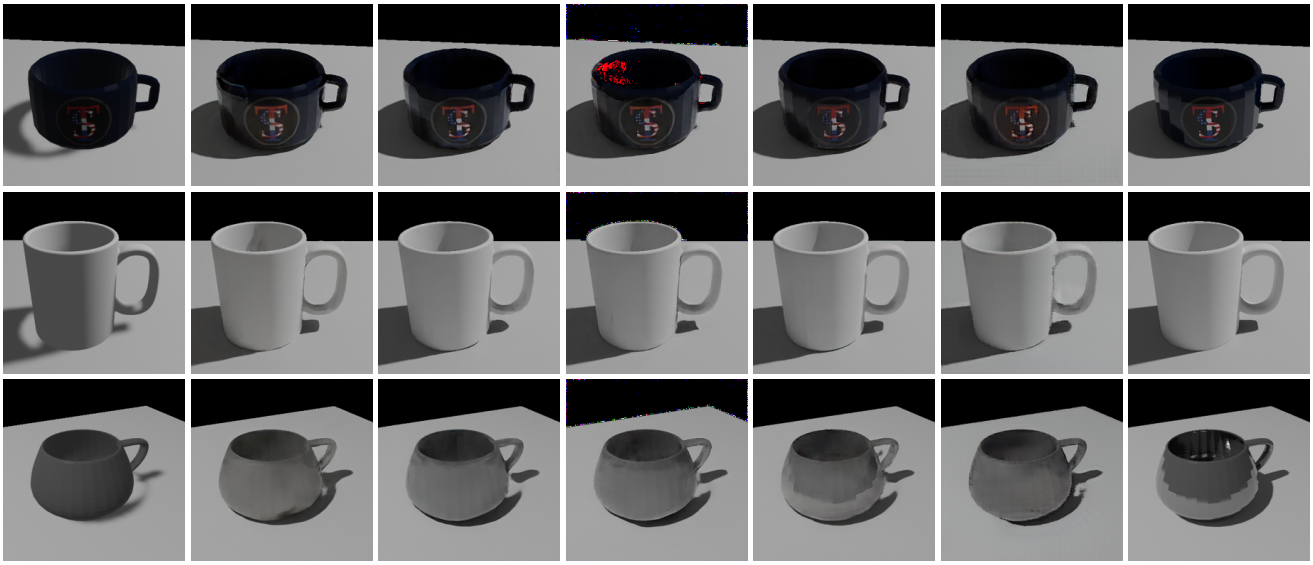
- **Base Model**: The base model does a decent job of predicting the ray-traced image. It fills in the imperfect shadows that are present in all the rasterized images and brightens the image just like the ray-traced version. However, there are some flaws. For example, in the top mug, the shadow of the handle is too small, and the reflection of the shadow on the mug is smooth compared to a sharp reflection in the ground truth. The shadow inside the second mug is curved as opposed to a straight shadow in the ground truth. Finally, the bottom mug still has a tiny hole in the shadow and does not have the curved reflections of the floor on the mug seen in the ground truth. We also noticed some loss of texture and warping of the mug's shape as well.

- **Depth Model**: Adding a depth map to the input improves the model's performance, as the size of the handle shadow in the top mug is larger, and the reflection of the shadow is sharper. The shadow inside the second mug is also more straight, though the hole in the bottom mug's shadow increased.

- **Depth + Skip Model**: Adding a skip connection led to an addition of random noise in the output image, as seen across all three mugs. The quality of the images looks close to the base model, but the presence of noise means that this model is unusable.

- **Depth + Normal Model**: Using a normal map in addition to the depth map helped the model greatly, as the handle shadow in the top mug, as well as the reflection of the shadow on the mug, now looks very similar to that of the ground truth. The shadow inside the second mug is very slightly curved, but it looks very similar to the ground truth. Finally, although the hole in the bottom mug's shadow is still present, the model learned to predict the reflection of the floor, albeit not perfectly.

- **Deeper Depth Model**: The additional depth of the model does not help with the output, as the images are similar, if not worse, than the original depth model. The model does not capture the benefits seen by the addition of a normal map. We theorize that one or both of the following factors explain the model's performance: (i) it suffers from an insufficient quantity of data, not a lack of depth, or (ii) the model could benefit from the

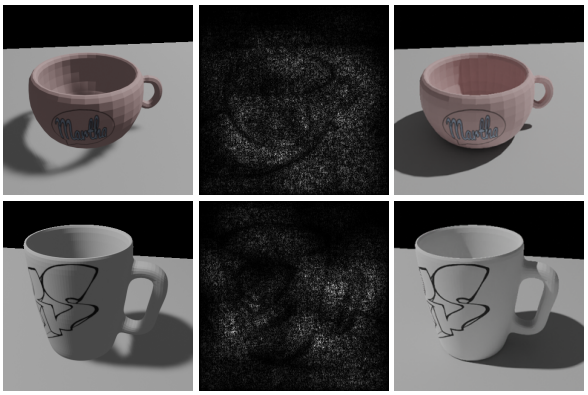


(a) Rasterized (b) Base (c) Depth (d) Depth+Skip (e) Depth+Normal (f) Deeper Depth (g) GT Ray-Traced

Figure 6. Comparison of models across 3 example mugs

additional decoder layers before the original decoder layers rather than after (Sec. 4.2) to preserve a different symmetry with respect to the skip connections.

#### 5.2.3 pix2pix Discriminator Saliency Map



(a) Eevee rendering (b) Discriminator saliency map (c) Cycles rendering

Figure 7. An example saliency map from the GAN discriminator compared to its respective Eevee and Cycles render

The saliency maps for the PatchGAN discriminator used in pix2pix show what features the filters the model is attending to in order to differentiate between real ray-traced images and fake generated ray-traced images. As shown in Figure 7, the saliency map focuses on the shadow of the mug as that is the part that the model is still struggling the most to fix. Similar to the saliency map in Figure 5, the model focuses on the edges of the mugs and the handle, specifically where the generative model is most likely to make changes in the lighting.

#### 5.2.4 Speedup of pix2pix:

The average inference time for all models was 0.08 seconds. Therefore, the average total time for our model to generate a ray-traced image is 0.37 seconds (0.21 to create the Eevee rendering, 0.04 to create a normal map, 0.04 to create a depth map, and 0.08 for model inference) while generating a Cycles image took 1.68 seconds. This is over a **4.5x speedup**.

## 6. Conclusion/Future Work

The differences in the output across our models, as well as the quantitative results, indicate that the best-performing model was the Depth + Normal model. We can therefore conclude that giving the model additional information, such as depth and normal maps, can greatly enhance the prediction performance. We hypothesize that giving the model even more information, such as the position of lights or the 3D scene itself, can further enhance performance. Furthermore, the 4.5x speedup of our model over the actual ray tracing performance is promising as it means that we can generate approximately similar quality images while drastically lowering rendering time. Additionally, even though the results of our models were impressive, they were not perfect. This indicates that there may be features that are impossible to predict without using physics-based ray tracing.

A future improvement to this project would consist of experimenting with various inputs like light position and object material to see which results in the best, or perhaps perfect, output. We could also add more convolutional layers after the extra skip connection, as that may help eliminate the discoloration artifacts. Furthermore, we may experiment with changing the depth differently at different model levels, which may give the model greater expressibility.

We may also try to produce the depth and normal maps using pix2pix instead of rendering them ahead of time, as doing so faster than the time it takes to render them would mean faster performance and simpler input, as the user would only have to input a single rasterized image as opposed to three images consisting of the rasterized image, depth map, and normal map.

## 7. Contributions/Acknowledgements

Daniel, Thomas, and Caroline made equal contributions towards this work, including but not limited to: (i) ideating with regard to the general approach and models, (ii) generating the dataset, (iii) running experiments, (iv) evaluating model outputs, (v) and writing the report. There was a slight variation in their points of emphasis, with Daniel leading the effort to create and train the CNN to distinguish ray-traced images from rasterized ones, Thomas leading the effort to set up the AWS infrastructure to train/test pix2pix, and Caroline leading the effort to evaluate the outputs of the experiments.

We consulted Jonathan Tremblay of Nvidia, and he thought that it would be possible to train a network such as pix2pix to generate ray-traced images from rasterized images. He suggested a dataset of one type of simple object—such as mugs—with a floor underneath, and we used code adapted from that used for his paper to generate our dataset.

Our pix2pix model was based on the code written by Jun-Yan Zhu and Taesung Park, and we consulted Jun-Yan Zhu for advice on modifying inputs to the model. We thank him for his time. The code is located at https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix, which is based on the paper [13].

Finally, many thanks to our TA, Yuan Gao, for his feedback throughout the project and suggestions on how to improve our experiment.

## References

[1] Louis Bavoil and Miguel Sainz. Screen space ambient occlusion. 10 2008. 2

[2] Anthony Paul Beug. Screen space reflection techniques. Master's thesis, University of Regina, 2020. 2

[3] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark, 2018. 2

[4] Marc-André Gardner, Kalyan Sunkavalli, Ersin Yumer, Xiaohui Shen, Emiliano Gambaretto, Christian Gagné, and Jean-François Lalonde. Learning to predict indoor illumination from a single image, 2017. 2

[5] Yanhui Guo, Xue Ke, Jie Ma, and Jun Zhang. A pipeline neural network for low-light image enhancement. *IEEE Access*, 7:13737–13744, 2019. 2

[6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018. 2, 4

[7] NVIDIA. Ray tracing. https://developer.nvidia.com/discover/ray-tracing, Accessed May 2023. 1

[8] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, ACM, 75-82 (2009)*, 02 2009. 2

[9] Li Tao, Chuang Zhu, Guoqing Xiang, Yuan Li, Huizhu Jia, and Xiaodong Xie. Llcnn: A convolutional neural network for low-light image enhancement. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2017. 2

[10] The Allen Institute for Artificial Intelligence. Objaverse. https://objaverse.allenai.org, 2023. 3

[11] Jonathan Tremblay, Moustafa Meshry, Alex Evans, Jan Kautz, Alexander Keller, Sameh Khamis, Thomas Müller, Charles Loop, Nathan Morrical, Koki Nagano, Towaki Takikawa, and Stan Birchfield. Rtmv: A ray-traced multi-view synthetic dataset for novel view synthesis, 2022. 2, 3

[12] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 5

[13] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020. 2, 8