ShadowNet: Deep Learning for Adding Shadows to a Scene

Tamer Mograbi Ameya Patil

May 21, 2019

Abstract

Rendering a scene with global illumination can be a time consuming task especially when we have complex objects and multiple light sources in the scene. Reducing this render time is our motivation behind applying deep learning methods for adding shadows to a scene. Given a rendered image of the scene without shadows, and a certain representation of the type and location of the light source, we aim to apply the image to image transformation paradigm to generate an image with shadows added. This problem entails prediction of occlusion of objects in the scene and also understanding the structure and orientation of the objects. In this project, we explore the applicability of the image to image transformation methods to the task of generating shadows in a scene, discuss the limitations and possible suggestions for improvements.

1 Introduction and Setup

Reducing the render time for global illumination for complex scenes having multiple objects and light sources is the motivation behind our project. However, we first tried to apply the image to image transformation method to simpler cases and ensured desirable results before moving on to more complex scenes. Accordingly, we started off with direct illumination and the following cases were tried and tested in the mentioned sequence:

1. Cbox scene with single diffuse sphere and point light at randomized locations
2. Cbox scene with single diffuse sphere with area light at randomized locations on the ceiling
3. Cbox scene with 100+ different objects at randomized locations and orientations with point light at randomized locations
4. Cbox scene with single chair object at randomized locations and orientations and point light at randomized locations
5. Same as above but with only locations randomized, and orientation fixed

Experiments 6 and 7 below were carried out with a different kind of input as explained in sections 3 and 4

6. Cbox scene with 10 different chair objects at randomized locations and orientations and point light at randomized locations

7. Cbox scene with single chair object at randomized locations and orientations and point light at randomized locations

The reason behind performing experiments in that particular order has been explained in the experiments section 4. For all the experiments, the object was confined within the bounds of the cbox so that we have maximum shadows being projected on the walls and ceiling and floor of the box. The position of the point light is always ensured to be outside the cbox and such that all the walls are fully and directly visible from the point light. For area light, the position is restricted only to the ceiling of the box. These restrictions ensure that we have ample light to create shadows.

Our code is available at shadowNet github

2 Network Details

We used the pix2pix network developed by Phillip et.al. [2] for the image to image transformation. Owing to the way the network was designed, supporting applicability to a wide variety of tasks, we could very easily use most of the pytorch implementation of their network except for a few changes in the data loader. We first provide a description of the image to image transformation network, the kind of input that it takes followed by the changes that we had to make to customize it for our purpose.

At the heart of the pix2pix network is the Conditional GAN, which generates an output image conditioned on an input image. As opposed to using the unstructured loss functions like L2 error or L1 error, the conditional GAN in pix2pix learns a structured loss. In unstructured loss, the individual pixels are treated independently, so the color assigned to one pixel has no effect on, or is not affected by the color assigned to some other pixel. This is not a desirable loss function for tasks where we want to retain the structure of the input image in the output image. A structured loss penalizes a joint group of pixels and thus helps in retaining structure. A traditional L1 loss was also used along with the structured GAN loss as it enforced the correctness of the low frequency components (high level structure) in the image. The GAN loss took care of the high frequency components (details) of the image. The final loss function used by the pix2pix network is thus designed as:

$$\mathbb{L} = \mathbb{E}_{x,y}[log(D(x,y))] + \mathbb{E}_{x,z}[log(1 - D(x, G(x,z)))] + \lambda \mathbb{E}_{x,y,z}[||y - G(x,z)||_1] \tag{1}$$

which the generator tries to minimize and discriminator tries to maximize.

A conditional GAN takes 2 inputs - one is the image which we want to transform to some other image - denoted as x in (1), and the other input is a random noise vector - denoted as z in (1). Accordingly, the discriminator also takes the input image, and the ground truth transformed image - denoted as y in (1) or the generated transformed image $G(x,z)$, to output a true or fake result.

Both the generator and discriminator networks have layers of the form Convolution-BatchNorm-ReLU. The generator network has a general form of the encoder-decoder network, where the input image is progressively downsampled until a bottleneck layer after which upsampling is performed till we reach the required shape for the output image. The downsampling step compresses a lot of information and this might cause some information to be ignored. However, retention of structure is a crucial requirement for image to image transformations since we are only trying to modify the original image, not completely change it. Skip connections are added between layers of the encoder half and decoder half having appropriate input and output dimensionality. These skip connections carry some information directly to the later layers and ensure that the structure of the image is retained across the transformation. These skip connections also give the network its name - 'U-net'

The discriminator network is designed keeping in mind that it needs to take care of the correctness of the high frequency components of the image. It does so by evaluating patches of images as being real or fake, instead of evaluating the entire image. By making the patch size smaller than the actual image size, and evaluating each patch, the discriminator network ensures that the high frequency components of the image are correctly created. This design gives the discriminator its name as PatchGAN. This design also helps in that the network has fewer parameters.

3 Data Generation

We used the Nori renderer to generate images for training, validation and testing. This saved us the time and efforts to learn using some other renderer and allowed us to spend more time on the main image to image transformation part. We had to make some additions to the Nori renderer as per our use cases which have been described below. The network takes an image as input, so we had to convert all our inputs to images and arrange them in the required format. Based on the experiments we ran, we had two types of inputs:

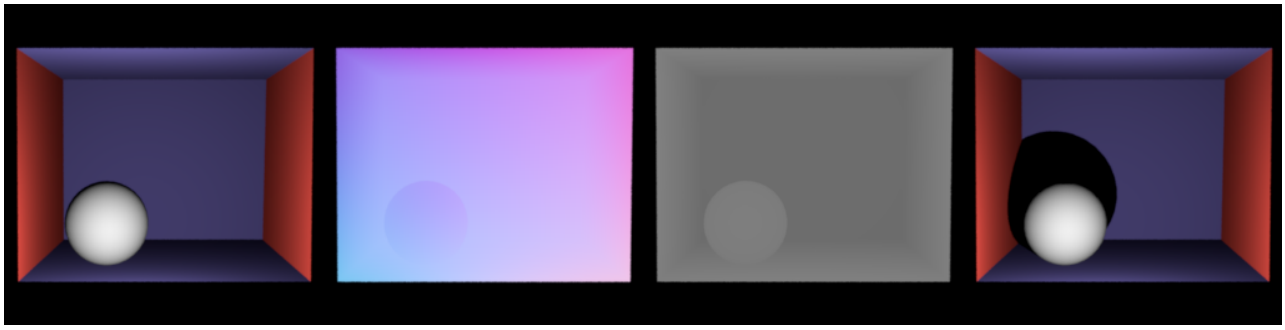
1. **4 image grid** (cases 1 to 5) has the image without shadows, the light map, the scene depth map and the ground truth image, refer Figure 1(a).

2. **11 image grid** (cases 6 and 7) has the 11 image grid consists of the image without shadows, the light map, 8 view depth maps and the ground truth image, refer Figure 1(b).

The meaning of these different types of images and how they were created, is explained in the next section.

3.1 Nori modifications

1. Point light map integrator:
   To represent the position of the light source in the scene as an image, we added a new integrator to Nori - lightMap integrator. For each hit point in the scene, the x, y and z components of the normalized shadow ray, was assigned as the r, g and b color values respectively, for the



(a) 4 image grid input for cases 1 to 5 - from left to right: noShadows, lightMap, depthMap, groundTruth

(b) 11 image grid input for cases 6 and 7 - from left to right: noShadows, lightMap, 8 views of depth map, groundTruth

Figure 1: Input images for the pix2pix network

corresponding pixels in the light map image. Values in the range of $[-1, 1]$ for the vector components were mapped to $[0, 1]$ to deal with negative values.

2. Area light map integrator:
   To represent the position of the area light source in the scene as an image, we added the area light map integrator. Since we had restricted the area light to be on the top ceiling, it was always in the field of view of the camera, and this integrator only rendered the area light. We also created another area light map integrator which created an image similar to the point light map integrator, this time the shadow ray was directed towards the center of the area light mesh. This allowed for the area light position to be randomized anywhere in 3D space with the same restrictions as the point light. However, we fell short on time before we could run experiments with it.

3. Depth map integrator:
   To give the network some indication about the relative position of objects in the scene, we added a depth map integrator. For each hit point in the scene, the inverse of the distance of the hit point from the camera origin was assigned as a grayscale value for its corresponding pixel in the image. We thus got a depth map with closer surfaces in the scene having a lighter color than the distant surfaces.

4. No shadows integrator:
   This is a slight modification of the simple direct illumination integrator in Nori. By removing the occlusion check, this integrator ensures that there are no shadows rendered in the image. This image acts as our main input. One important thing to note here is that **surfaces which do**

not receive direct light are still rendered black. Only the occlusion check was removed to avoid shadows, since that is what we want the network to learn. By having the surfaces not receiving direct light, rendered as black, our understanding is that this gives the network additional cues about what is the structure and orientation of the object.

3.2 Dataset

For the initial stage of getting the network setup, we used the 3D meshes provided in the Nori assignments. Later, we used the IKEA furniture dataset which had 3D meshes in .obj format. of chairs, tables, desks, bed frames and wardrobes. ShapeNet dataset was our first choice for 3D objects but for some reason, the images for ShapeNet objects were not being rendered in the Nori renderer. We did not spend much time trying to debug this issue and moved on to the IKEA dataset.

3.3 Script for creating randomized scenes

To randomize the scenes, we had to change the locations of the vertices of the objects in the .obj files, so that the objects are all **located inside the cbox** and **fit completely inside it**. We first attempted to compute the bounds of the positions where we can translate the object to, so that it satisfies the two constraints, by hit and trial. This was tricky since we were also repurposing the .xml files from the Nori assignments, which had the camera located at and oriented along axes for which it was inconvenient to compute the bounds manually. We got around this issue by changing the camera parameters to:

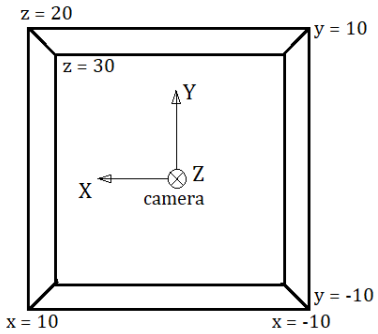


Figure 2: Cbox and camera frame of reference in world coordinate system after simplification

- from: $(0, 0, 0)$
- to : $(0, 0, 1)$
- up: $(0, 1, 0)$
- field of view: 60^o

Once this was done, we manually modified the cbox walls to span a volume from $x \in [-10, 10]$, $y \in [-10, 10]$ and $z \in [20, 30]$. This simplified things as we had now set our own camera frame of reference. We then performed the following steps for every .obj file to generate randomized scenes:

1. Compute the center of mass - COM of the vertices in the mesh
2. Center the object at the origin by subtracting mean from all the vertices

3. Normalize the vertices to have unit standard deviation, to scale them to fit them in the cbox

4. Generate random rotation matrix and apply it to the vertices

5. Translate the vertices at random locations such that they completely fit inside the cbox

The obj file was re-written after the above steps. The same steps were followed for randomizing the area light mesh locations. For point light, the x, y and z coordinates were randomly chosen within the constraints for ensuring all the surfaces received direct light. The .xml files were modified with the new light position.

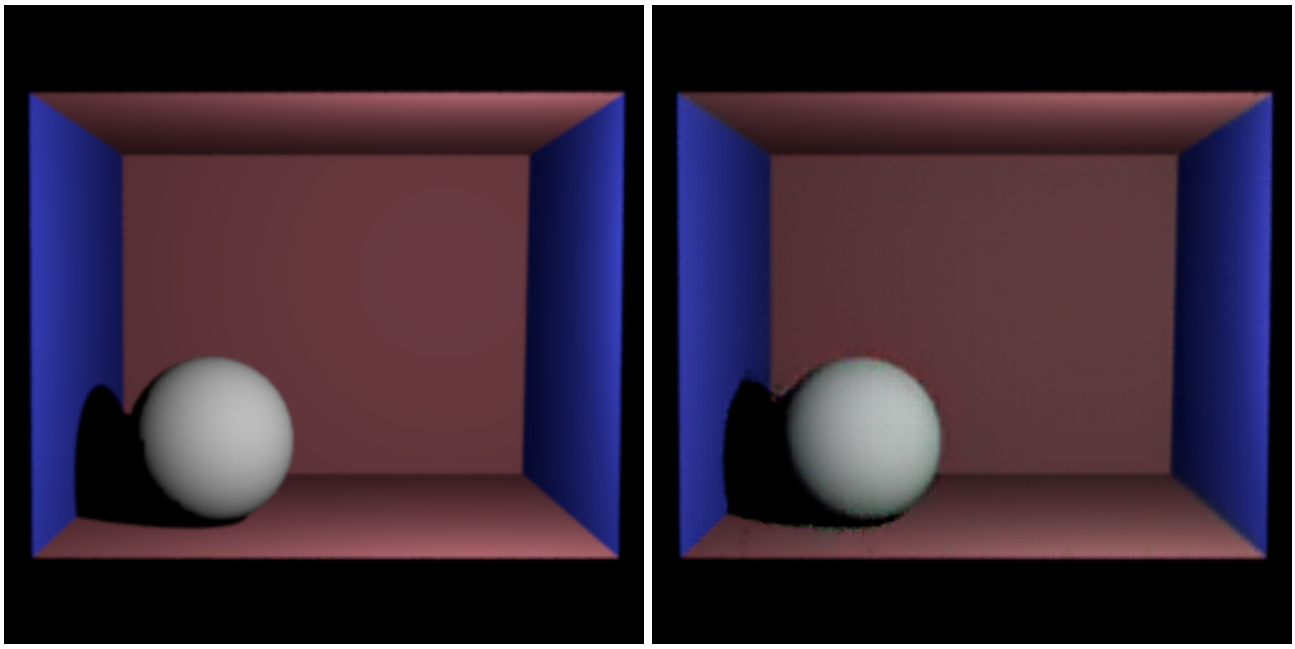
For every randomized version of the object and light position, all the integrators mentioned in section 3.1 were run and corresponding images were rendered. The 8 view depth maps were generated by keeping only the object (no cbox walls) and rotating the object with fixed angles of $\pm 45^o$ about X axis and, $\pm 45^o$ and $\pm 135^o$ about the Y axis. These images were also aligned in the required format as shown in Figure 1

4 Experiments and Results

Since the original pix2pix network only accepted single image input and we had multiple image inputs, we modified the dataloader section of the pix2pix network to break down the 4 or 11 image grid and stack them along the 3rd dimension as channels. Further, we also added a new version of the generator network which could take in 512x512 images on similar lines as the existing network. Shadows in complex scenes can be very intricate and so to ensure that the images have good enough resolution to capture those intricacies, we made this addition. We ran the first 2 experiments on 256x256 images and the last 5 experiments on 512x512 images where the shadows started becoming more intricate. In all the cases, the network was trained for 200 epochs. The PatchGAN discriminator was using patches of size 70x70

4.1 Scene 1: Cbox with single diffuse sphere and point light

Since this was only meant to get the network working, it was trained on 450 different scene configurations having randomized point light locations. 4 image grids were created where each image was of size 256x256. The testing results are shown in Figure 3. The results were as expected.

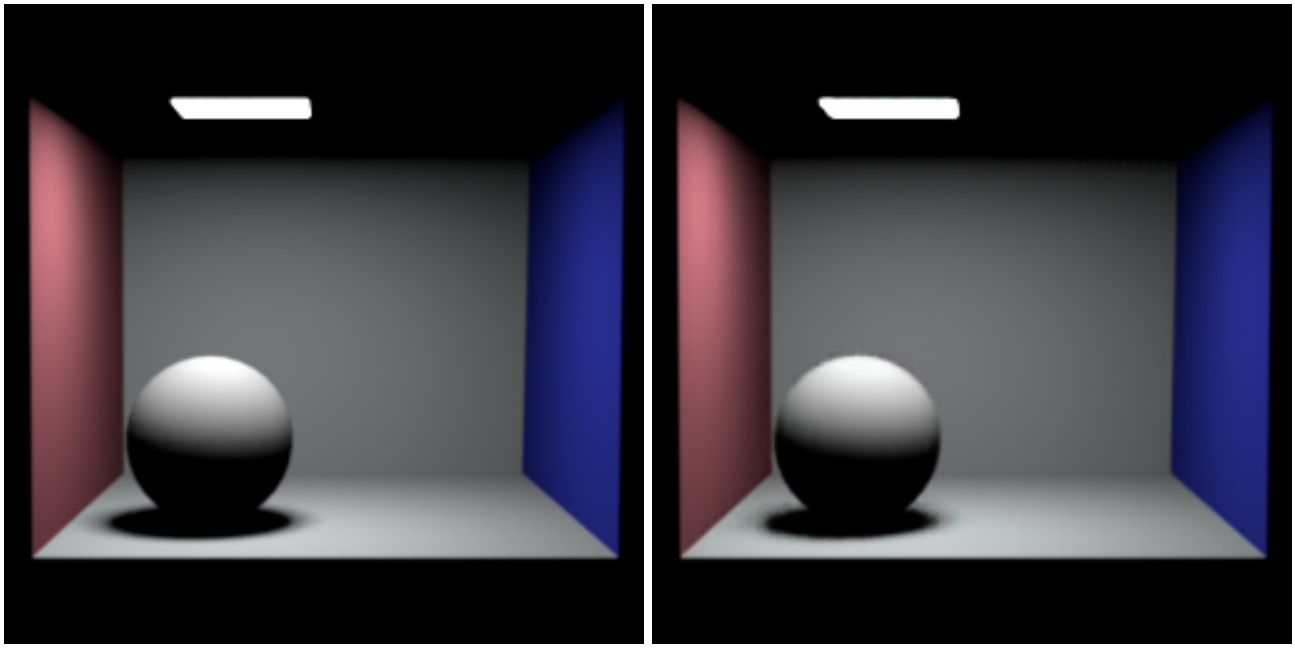


(a) Ground truth image on the left, image generated by the network on the right

Figure 3: Test results for the pix2pix network for Scene 1 - single diffuse sphere and point light at randomized locations, using the 4 image grid input

4.2 Scene 2: Cbox with single diffuse sphere and area light

The next step was to check if the network is able to learn rendering soft shadows as well. The network was trained on 450 different scene configurations. 4 image grids were created with each image being 256x256. The test results are as expected and are shown in Figure 4

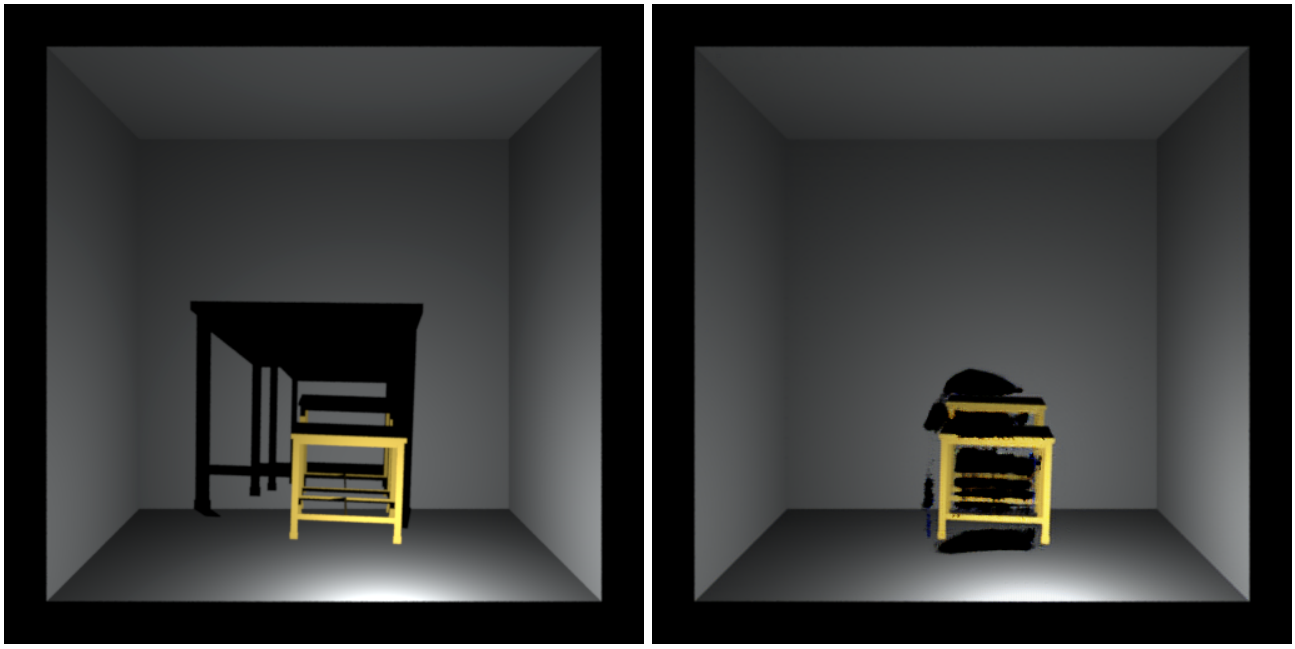


(a) Ground truth image on the left, image generated by the network on the right

Figure 4: Test results for the pix2pix network for Scene 2 - single diffuse sphere with area light at randomized locations, using the 4 image grid input

4.3 Scene 3: Cbox with different objects and point light

The next experiment we performed was with random objects from the IKEA dataset. We first chose the point light since we were expecting the shadows to be intricate in this case. We trained the network on 1000 images having the objects and point light at random locations and orientations, Image resolution was set to 512x512 in view of the intricate shadows. One of the test results for a table, is shown in Figure 5



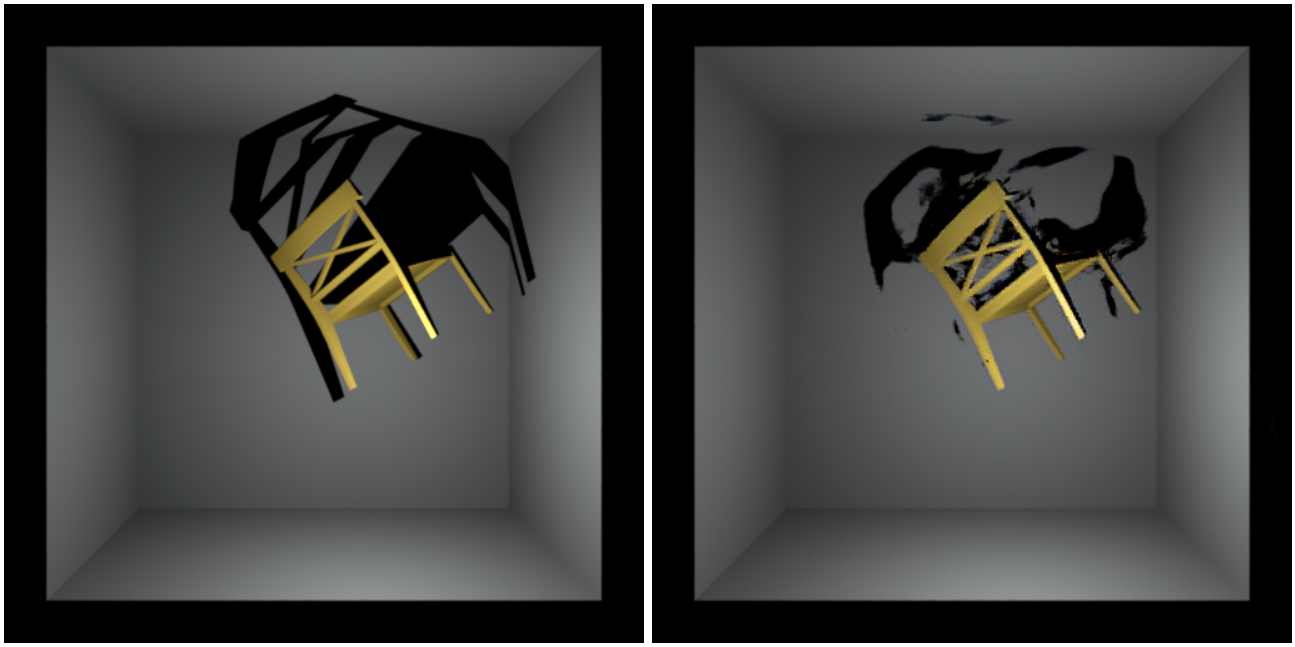
(a) Ground truth image on the left, image generated by the network on the right

Figure 5: Test results for the pix2pix network for Scene 3. 100+ different objects with point light at randomized locations and orientations, using the 4 image grid input

In this case, the network always gave bad results. It probably was unable to understand the object structure and orientation. From the test results, it shows that the network did figure out the region of where the shadow should be but not the shape of it.

4.4 Scene 4: Cbox with single chair object and point light (4 image grid input)

Our guess for the failure of the previous experiment was that the network was unable to learn the structure of the object since there were 100+ different objects. So in this experiment we decided to train the network with randomized locations and rotations of a single chair object from the IKEA dataset, in the hope that it would learn the structure. This makes this experiment similar to 4.1. 400 images of 512x512 size were created for training. Even for this experiment, we found out that the results were not as expected, as shown in Figure 6

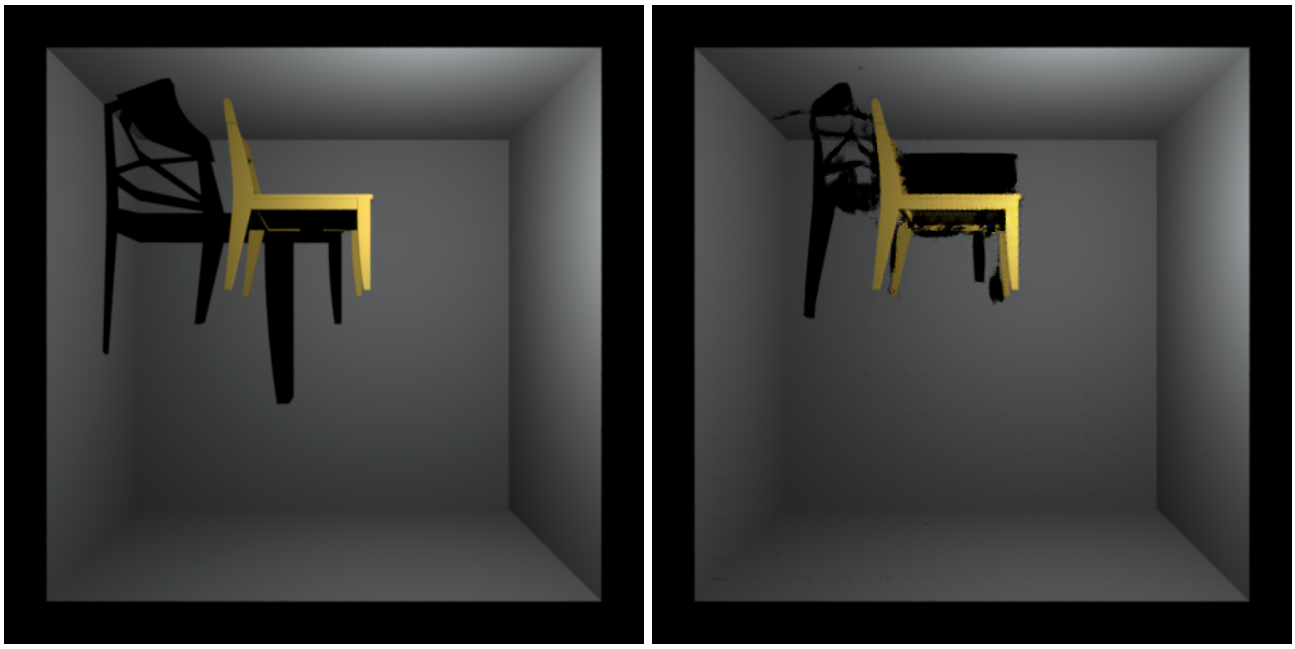


(a) Ground truth image on the left, image generated by the network on the right

Figure 6: Test results for the pix2pix network for Scene 4 - single chair with point light at randomized locations and orientations, using 4 image grid input

4.5 Scene 5: Cbox with single chair object, no rotations and point light

In this experiment we tried eliminating random rotations completely, and kept only translations just to check what effect does it have on the results. We generated 350 images of size 512x512 for training. The results as shown in Figure 7, were still not satisfactory.

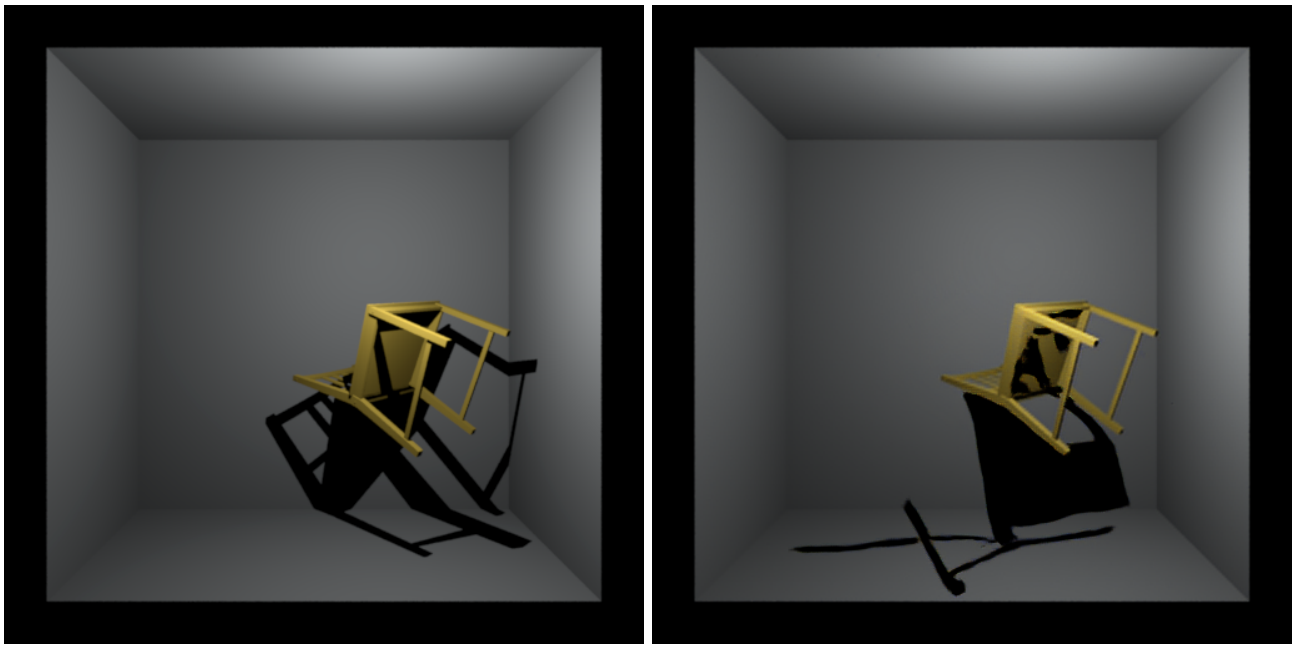


(a) Ground truth image on the left, image generated by the network on the right

Figure 7: Test results for the pix2pix network for Scene 5 - single chair with no rotations, only translation, using 4 image grid input

4.6 Scene 6: Cbox with 10 chairs and point light (11 image grid input)

Our understanding from the failures of the previous experiments on randomized objects was that the network was failing to understand the structure of the object. So in this experiment we decided to use the 8 view depth maps in place of the single depth map that was being used in all the previous experiments. This approach of representing structure of 3D objects was referred from [1]. 1500 such images of 10 different chairs were used, each of size 512x512. 10 images were input to the network stacked along the channels - the no shadows image, the light map and the 8 view depth maps, refer Figure 1(b). The results are as shown in Figure 8

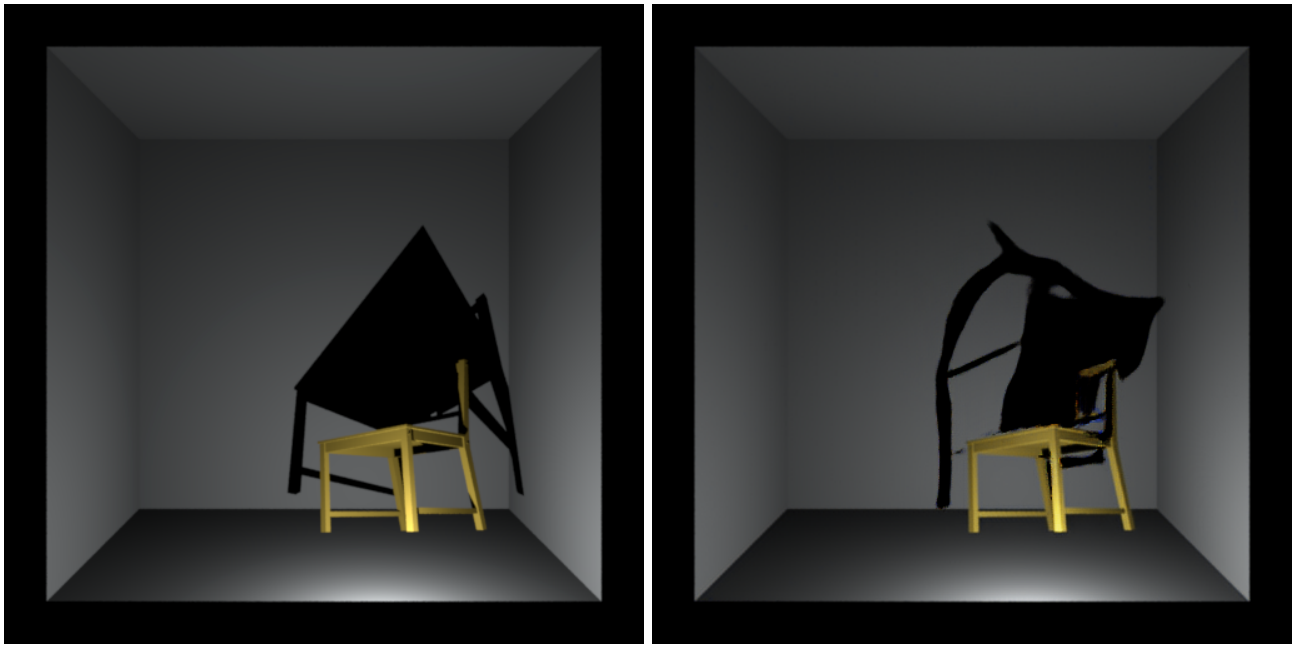


(a) Ground truth image on the left, image generated by the network on the right

Figure 8: Test results for the pix2pix network for Scene 6 - 10 chairs and point light at randomized locations and orientations, using the 11 image grid input

4.7 Scene 7: Cbox with 1 chair and point light (11 image grid input)

Even after training the network on only chairs, it was not able to understand the structure and orientation of a chair. So the next and the last experiment that we performed was of training the network on only one chair, just to make sure that the issue is really in the network architecture and not in the input data. 400 such images of a single chair with randomized locations and orientations were created, each of size 512x512, along with the 8 view depth maps and the light map. The results are as shown in Figure 9



(a) Ground truth image on the left, image generated by the network on the right

Figure 9: Test results for the pix2pix network for Scene 7 - single chair and point light at randomized locations and orientations, using the 11 image grid input

5 Problems faced

1. Data wrangling:
   Many of the meshes in the dataset had surfaces which just did not reflect any light and were rendered as dark surfaces. We tried modifying the vertex normals along with the geometric vertices in the process described in section 3.3, in the hope of resolving this issue, but to no effect. So we had to manually go through the dataset and remove any such object which was giving us faulty images. These images could be acting as bad data points and messing up the learning of shadows for the network.

2. Area light map:
   Although we ran only one experiment with area light source, we could not come up with a different light map representation for point lights and area lights for the case where area light could be outside the field of view of the camera. A different representation would be needed if the network were to understand the light source type and size by just looking at the light map.

3. Confusing training data:
   Since experiments were run only on direct illumination, parts of the object which were not facing the light source were also rendered as black, the table top in Figure 5 serves as a good example. We saw this issue after we shifted to the IKEA dataset. This can potentially confuse

the network as to what is shadow and what is the object part exactly. Global illumination could have avoided this confusion, but the high data generation time for global illumination limited the variations that we could try out.

6 Conclusions and Future Scope

The network only seems to be able to add shadows to a simple scene comprising of a simple shape with point light or area light. Things start falling apart when complex scenes are used, owing to understanding the complexity of the structure and orientation of the object in the scene from 2D images. The location of the rendered shadow was close to the ground truth, but the shape of the shadow was way off, indicating that the light map was doing its job, and more work needs to be done on learning the structure and orientation of the objects in the scene. We were able to come up with some more possible changes that we could try out next:

1. Separate networks to learn structure and occlusion:
   Learning structure and occlusion are the 2 crucial parts of this task. We could accordingly employ an ensemble model of networks where one network learns the structure from multiple 2D views of the 3D object and the other network learns only the occlusion. The outputs from these two networks can together determine the final output.

2. No latent vector:
   The conditional GAN takes as input a random latent vector and an image conditioned on which we want our output to be generated. The role of the latent vector here is to produce new varieties of output for the same basic structure, i.e. to make the output generation more stochastic and less deterministic. Our use case however requires that the output be deterministic. We could in that case, do away with the latent vector and check how the network performs. However this is not trivial in the pix2pix framework as the latent vector is not input at the very first layer of the generator, along with the input image. It is input at some intermediate layer because, that way it was found to introduce better stochasticity in the output.

References

[1] Hu, T., Han, Z., Shrivastava, A., and Zwicker, M. Render4completion: Synthesizing multi-view depth maps for 3d shape completion. *CoRR abs/1904.08366* (2019).

[2] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* (2017).