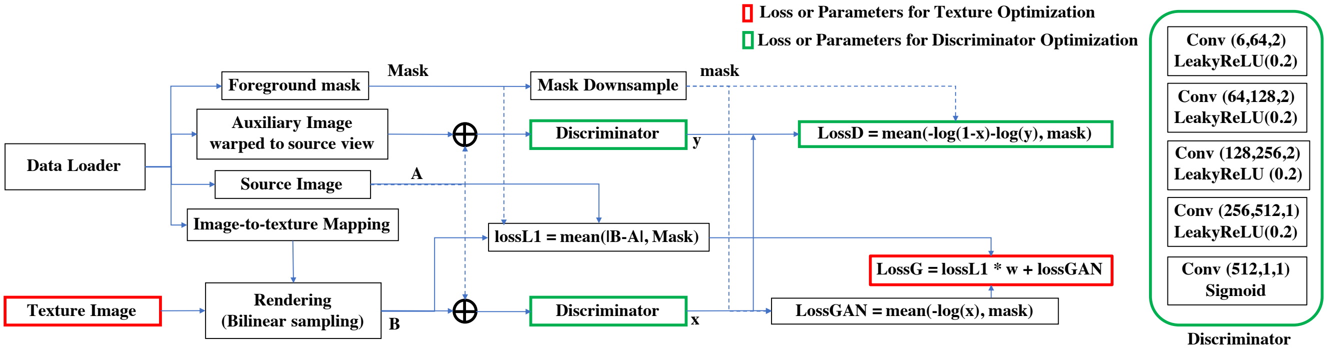

🟥 Loss or Parameters for Texture Optimization
🟩 Loss or Parameters for Discriminator Optimization
Data Loader
Foreground mask
Mask
Mask Downsample
mask
Auxiliary Image warped to source view
Discriminator
y
LossD = mean(-log(1-x)-log(y), mask)
Source Image
A
Image-to-texture Mapping
lossL1 = mean(|B-A|, Mask)
LossG = lossL1 * w + lossGAN
Texture Image
Rendering (Bilinear sampling)
B
Discriminator
x
LossGAN = mean(-log(x), mask)
Conv (6,64,2) LeakyReLU(0.2)
Conv (64,128,2) LeakyReLU(0.2)
Conv (128,256,2) LeakyReLU (0.2)
Conv (256,512,1) LeakyReLU(0.2)
Conv (512,1,1) Sigmoid
Discriminator

Figure 13. Detailed description of the optimization process.

Supplemental

A. Implementation details

A.1. Data Loader During Optimization

Generating real and fake examples are two critical steps in our optimization process.

To achieve this, we provide three images for each input view as shown in Figure 12. For each view, we provide the color and the depth image from the device, as shown in the first two rows. The background pixels can be removed simply by deciding whether the corresponding rays are intersection the reconstructed mesh. Additionally, we provide the view-to-texture mapping for the differentiable rendering of the texture image.

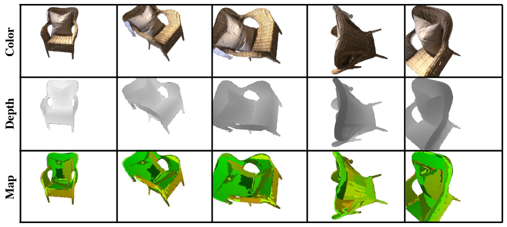


Figure 12. For each view of the scan, we cache the corresponding color and depth images. In settings where depth data is unavailable, we render depth maps from the target geometry. Additionally, we pre-compute the image-to-texture map for differentiable rendering.

Figure 13 shows the details of our optimization process. For each iteration, the data loader randomly selects a pair of related views as the source and the auxiliary, and we feed the network with a foreground mask containing the 3D scan, the real example synthesized by reprojecting the auxiliary image to the source view, the source image and the image-to-texture mapping. In the texture optimization stage, we render the target texture to source view based on image-to-mapping with a differentiable bilinear sampling as B. It is combined with the source image A as condition, sent to the discriminator to derive the prediction x. x is used to compute the adversarial loss as "lossGAN". We additionally compute the L1 loss between A and B as "L1", and linearly combine "lossGAN" and "L1" with an exponentially decayed w as "lossG", which is the objective function for optimizing the texture image. For every 1000 steps, we exponentially decay the w by a factor of 0.8. In the discriminator optimization stage, the real example is combined with source image A and sent to discriminator to derive the prediction y. x from the fake example and y from the real example are combined to compute the object adversarial loss "lossD" for discriminator optimization.

The discriminator architecture consists of 5 convolutional blocks (figure 13). Conv(x,y,z) represents a 4x4 convolution with padding as 1, input channel as x, output channel as y and stride as z. Each convolution block is followed with a gate function where the first four are leakyReLUs and the last is sigmoid.

A.2. Details for Synthetic 3D Data Generation

We studied the behavior of our approach given the inaccurate camera pose or geometry. Camera perturbation is achieved by adding uniformly distributed noises to each dimension of the translation ranging from $[-e_t, e_t]$ and rotation as euler angle ranging from $[-e_a, e_a]$. To simulate geometry errors, we randomly generate a scalar for each vertex following the uniform distribution ranging from $[-e_g, e_g]$. Then, we apply 3 steps of Laplacian smooth to the scalars. We move the vertices along their normal directions with the distance specified by these scalars. We compare our method with different approaches for all selected ShapeNet objects with different amount errors. For camera errors, we set $e_t = 0.01 * 1.5^n$ ($n \in \{1.5, 2, 2.5, 3, 3.5, 4, 4.5\}$) and $e_a = 5°$. For geometry errors, we set $e_g = 0.02 * 1.5^n$ with $n \in \{1.5, 2, 2.5, 3, 3.5, 4, 4.5\}$.

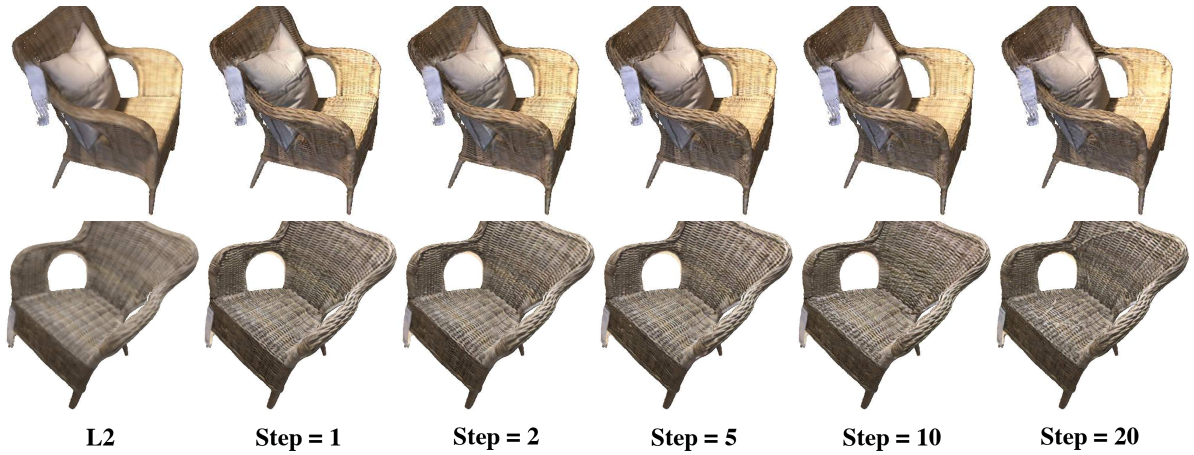


Figure 14. Ablation study on view density required for training. The two video sequences are with 426 and 244 frames respectively. The first column shows the average of frames under L2 loss. We sample the sequence by uniformly pick frames every k steps, with $k \in \{1,2,5,10,20\}$ in the remaining five columns.

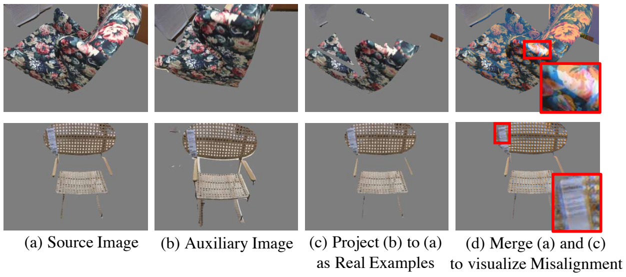

(a) Source Image (b) Auxiliary Image (c) Project (b) to (a) as Real Examples (d) Merge (a) and (c) to visualize Misalignment

Figure 15. Real examples with Misalignment. (a) is the source image. (b) is an auxiliary image. In (c), we project (b) to the geometry and render to (a), which is a misaligned version of (a). (d) visualizes the misalignment by overlaying (a) and (c).

B. Misalignment in the Data

Figure 15 shows an real example of the misaligned version of source views that we created.

C. Sparsity of Views

We run our algorithm on scans with different number of frames to study the behavior and the robustness of our algorithm under different level of view sparsity, as shown in Figure 14. The two video sequences are with 426 and 244 frames respectively. The first column shows the average of frames under L2 loss. We sample the sequence by uniformly pick frames every k steps, with $k \in \{1,2,5,10,20\}$ in the remaining five columns. We notice that our algorithm produces appealing results if number of frames is larger than 25, but starts to show artifact patterns under this number. We believe the reason is that with very sparse views, there are not enough patches for learning a good misalignment tolerant metric. We believe this can be addressed by data augmentation with virtual camera perturbations.

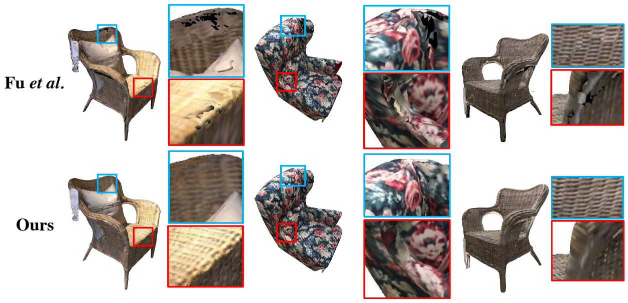
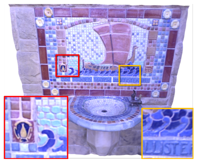


Figure 16. Comparison with [13] that develops based on [32].

D. Additional Comparisons



Our method can deal with standard texture optimization dataset well as shown in the example of the fountain scan. Our experiments are aimed to showcase more challenging scenarios with complex scene geometry and lighting, as well as approximate surface reconstruction and alignment.

We provide additional comparisons between our method and Fu *et al.* [13] that develops based on [32]. View selection-based method yields inconsistent boundaries, while our method generates consistent texture.

E. Additional Results

We provide a video called “supp/video.mp4” that contains the explanation of our approach and part of video and image results. We additionally provide the visualization of the full real dataset in our experiments with 200 renderings comparing to different methods for objects, scenes and CAD models. Please check ”supp/*.html” for details.