– Supplementary Material –
Learning an Isometric Surface Parameterization for Texture Unwrapping

Sagnik Das[1], Ke Ma[2], Zhixin Shu[3], and Dimitris Samaras[1]

[1] Stony Brook University, Stony Brook NY 11790, USA
{sadas}@cs.stonybrook.edu
[2] Snap Inc.
[3] Adobe Research

Overview

In this supplementary material we include the following sections:

1. High-resolution results for arbitrary surface texture editing
2. High-resolution results for document texture editing
3. Video with additional results
4. More qualitative comparison with DewarpNet [4] on synthetic evaluation set
5. Qualitative comparison with [4] for different types of real documents
6. Qualitative comparison with [16] on their test-set
7. Usefulness of L_{uv}
8. Details of weighting function used in L_z
9. Training details of the UV prior network
10. Initializing S and F_z
11. Unwarping and texture editing details
12. Pre-processing details for the real scenes
13. Detailed ablation figure
14. Limitations
15. Example of a failure case
16. Training time

1 High-resolution results for arbitrary surface texture editing

In Fig. 1, 2 and 3 we show the examples of editing arbitrary surface texture. Furthermore, in Fig. 4, 5 we show examples of face [10] texture unwrapping and editing. These examples show that our learned F_{uv} prior and the proposed method works beyond documents as long as the isometry assumption is not strongly violated.

2 High-resolution results for document texture editing

In Fig. 6, 7 we show the examples of texture editing in higher resolution.

Fig. 1. Example of texture edited images rendered from different views. Note the perspective changes and deformation on the edited texture due to the surface. The input foreground mask is shown using dashed yellow polygon.

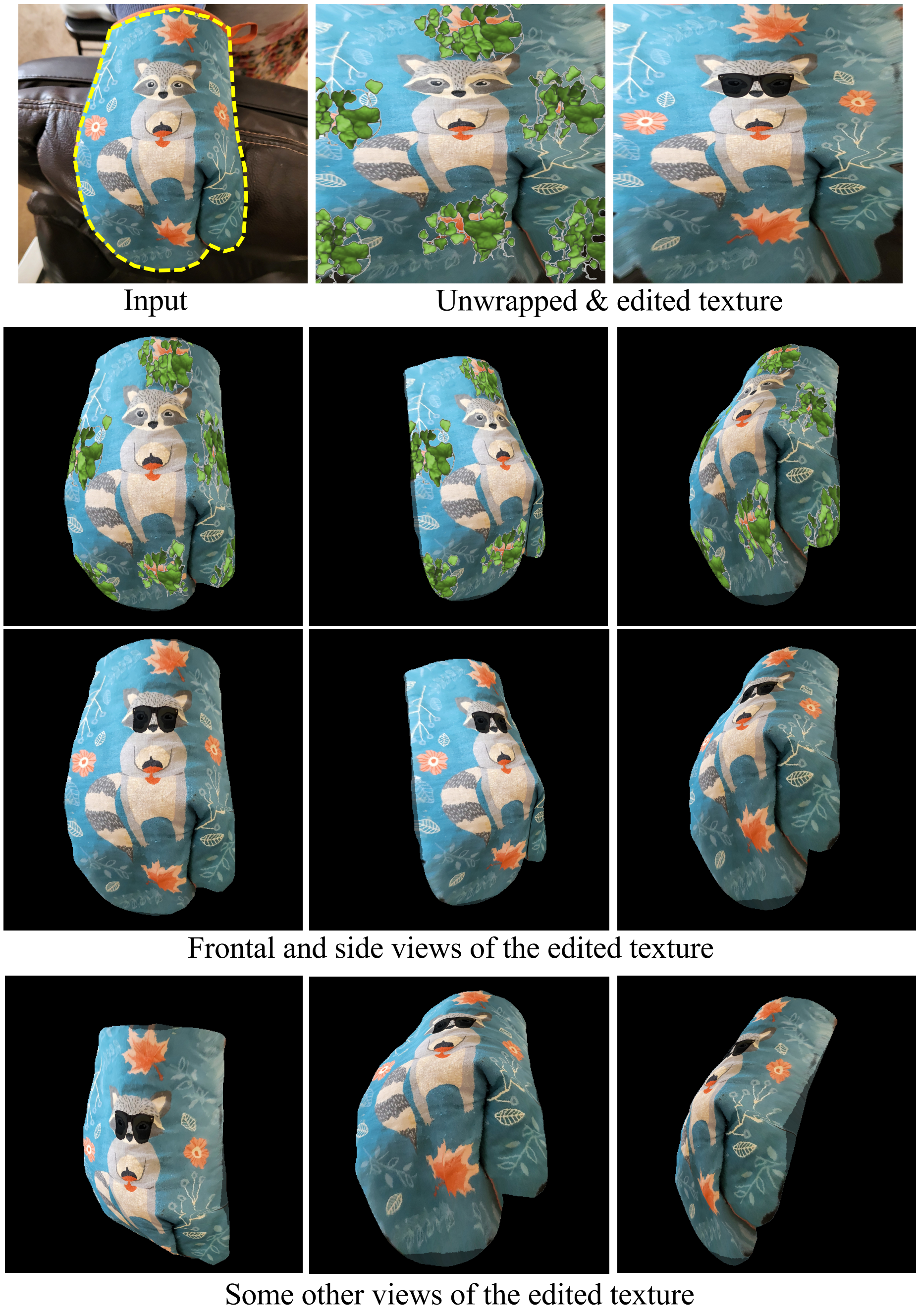


Fig. 2. Example of texture edited images rendered from different views. Note the perspective changes and deformation on the edited texture due to the surface. The input foreground mask is shown using dashed yellow polygon.

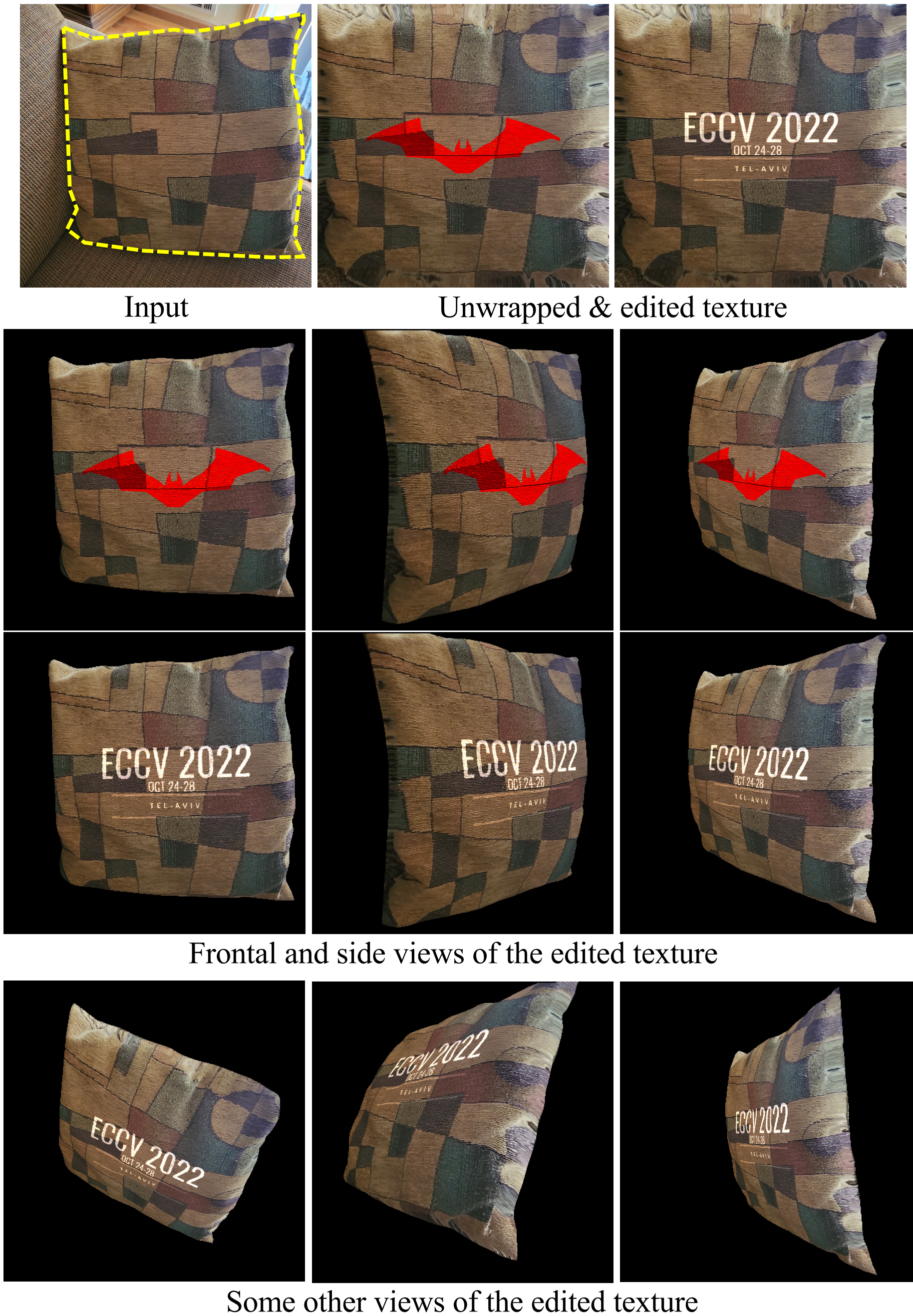


Fig. 3. Example of texture edited images rendered from different views. Note the perspective changes and deformation on the edited texture due to the surface. The input foreground mask is shown using dashed yellow polygon.

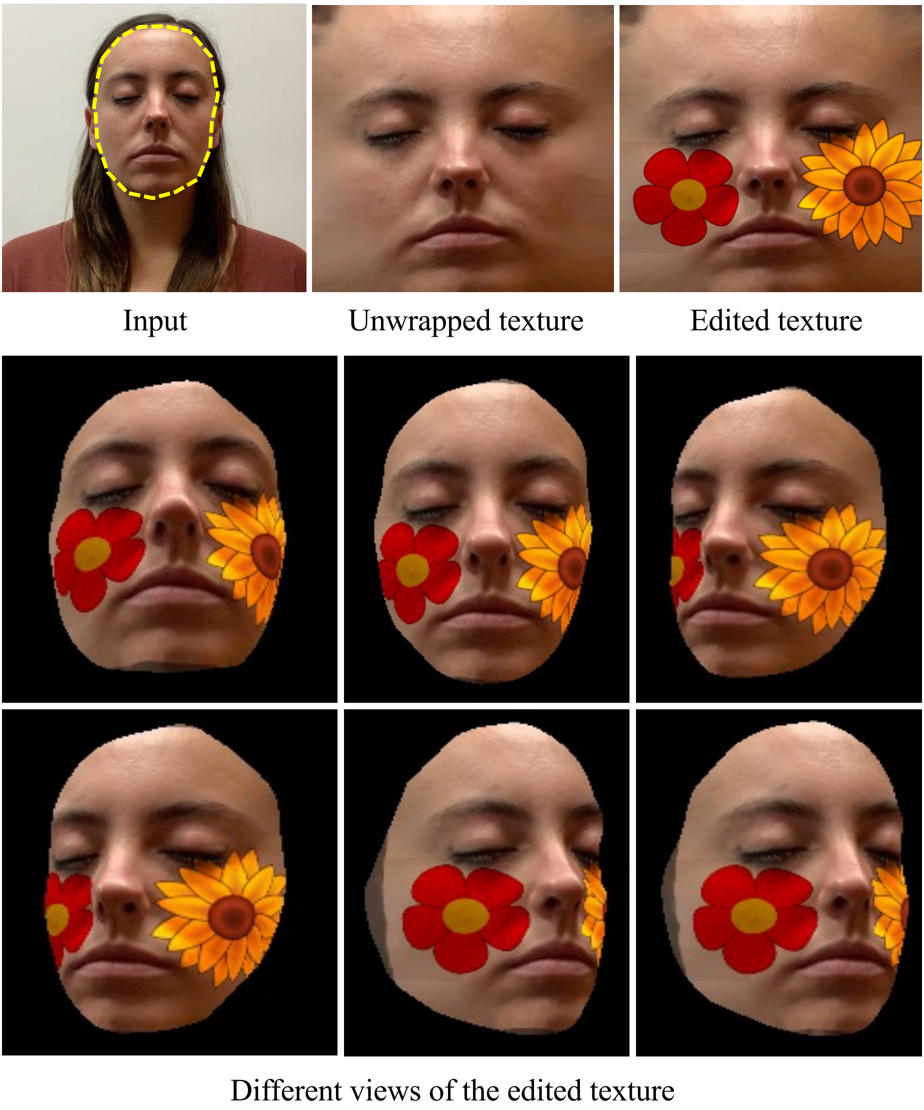


Fig. 4. Example of texture edited faces [10] rendered from different views. Note the perspective changes and deformation on the edited texture due to the surface. The input foreground mask is shown using dashed yellow polygon.

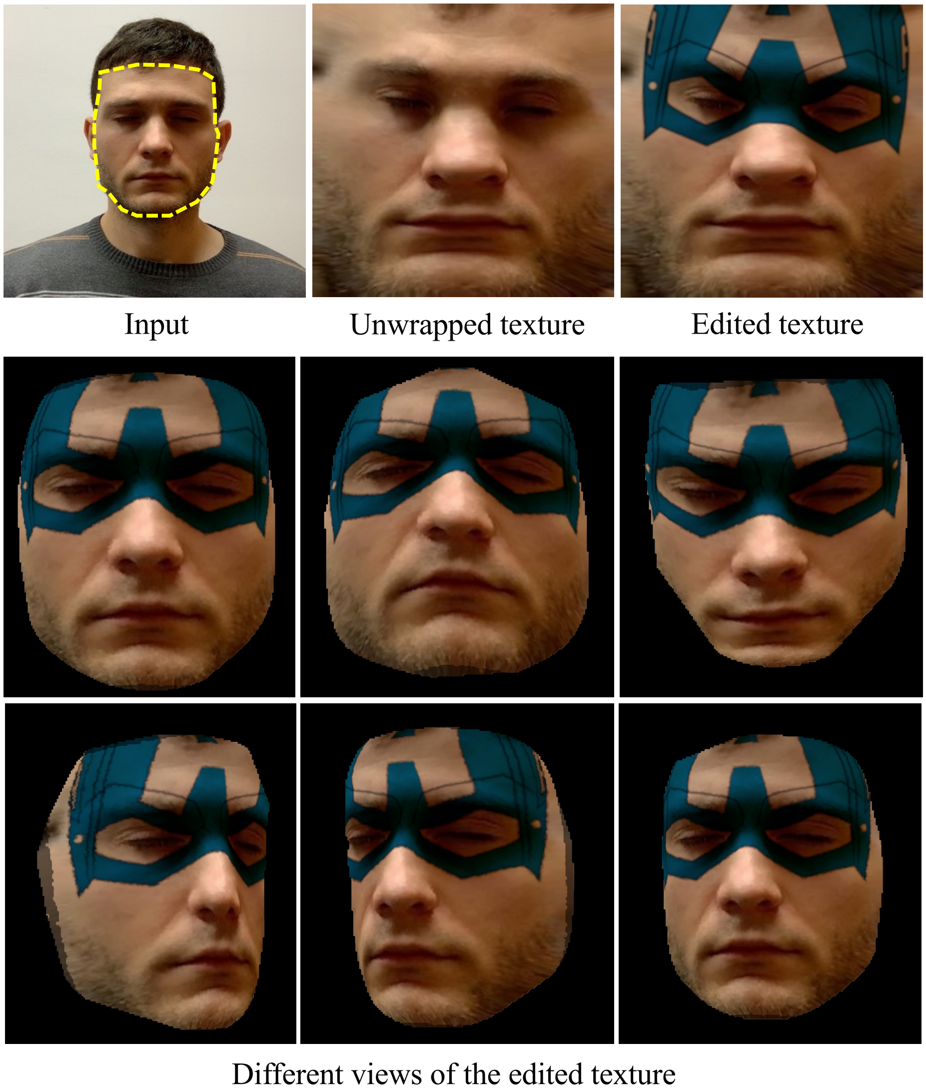


Fig. 5. Example of texture edited faces [10] rendered from different views. Note the perspective changes and deformation on the edited texture due to the surface. The input foreground mask is shown using dashed yellow polygon.

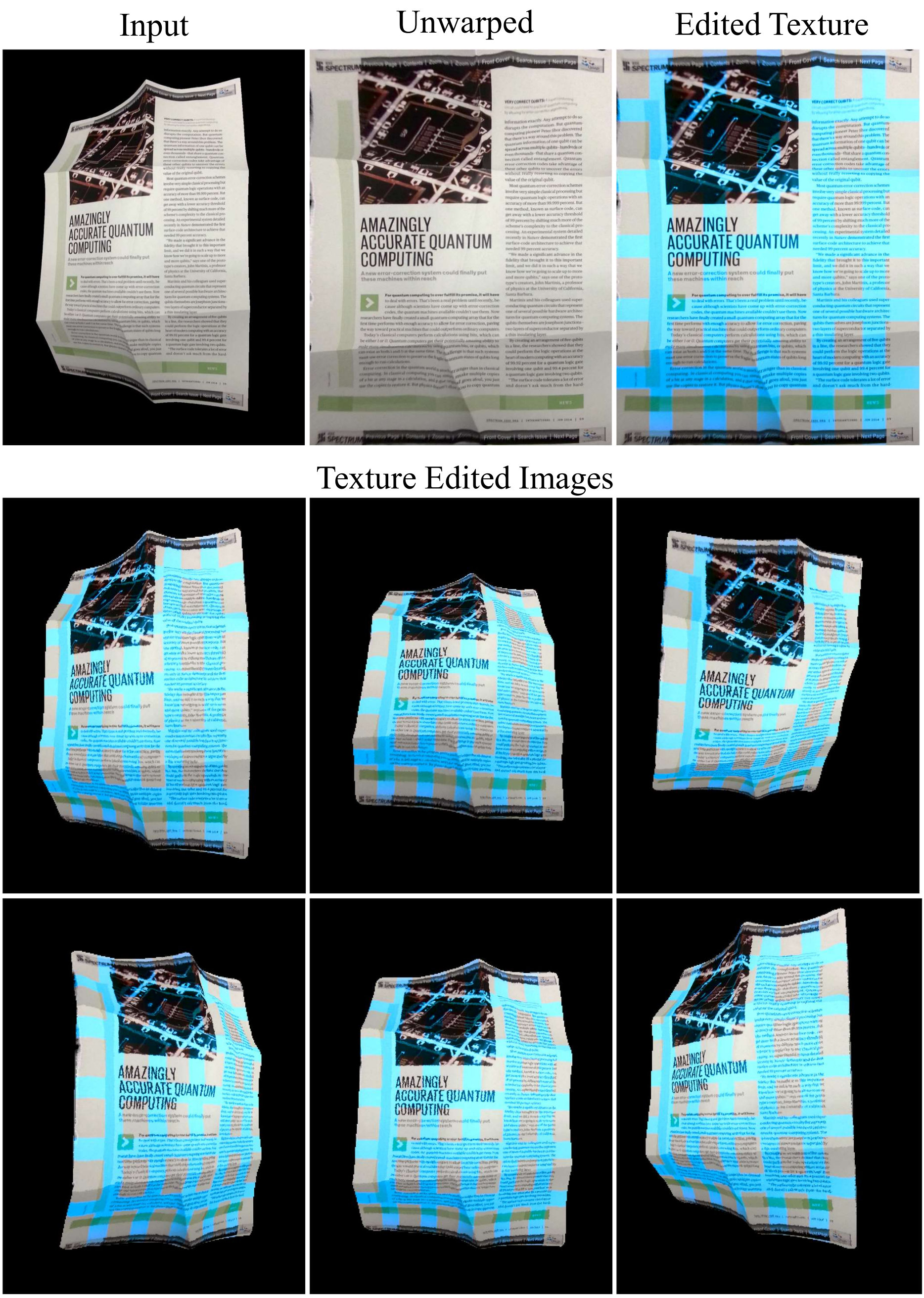


Fig. 6. Example of texture edited images from different views. Note the perspective changes and deformation on the edited texture due to the complex shape of the paper.

Fig. 7. Example of texture edited images from different views. Note the perspective changes and deformation on the edited texture due to the complex shape of the paper.

3 Video with additional results

We include a video (3394-supp.mp4) to demonstrate the quality of our texture editing results. It includes continuous view of the edited textures from different camera perspectives.

4 More qualitative comparison with DewarpNet [4] on synthetic evaluation set

In Fig. 8, 9 we show more qualitative comparison with DewarpNet [4] on unwarping frontal view of a document. For a better illustrative comparison we also show qualitative results of the 4 best (lowest LD) unwarped views using [4] in Fig.10, and 11. Clearly in all of the cases we achieve better or comparative results. Furthermore, we can see that it is hard to predict which view will perform best for [4], and results vary significantly even if the views are reasonably frontal. Comparatively, being a multi-view method, our approach produces more consistent unwarping across all views.

5 Qualitative comparison with [4] for different types of real documents

In Fig. 12, 13, 14, and 15, we show qualitative unwarping result for four different type of documents, e.g. book, receipt, flyer, and magazine. In all the views our method shows consistent and good quality unwarping results.

6 Qualitative comparison with [16]

In Fig. 16, 17 we provide a qualitative comparison with 5 publicly available images from [16]. The results are competitive and often produce better unwarping. Quantitative numbers couldn't be reported because the high-res/original unwarped results are not publicly available.

7 Usefulness of L_{uv}

In section 3.3 of the main submission, we define L_{uv} (Eq. 8) to prevent non-uniform mapping between the 3D and the UV domain. Specifically, we constrain the output of F_{uv} to be $\sim \mathcal{U}(0,1)$ using L_{uv}. Without L_{uv}, F_{uv} is prone to produce a mapping $\sim \mathcal{U}(a,b)$ where $a > 0$ or $b < 1$. Consequently, F_z also learns an incorrect mapping between the texture and the 3D domain. As a result, the unwarped texture gets stretched or squeezed. We demonstrate two such examples in Fig. 18.

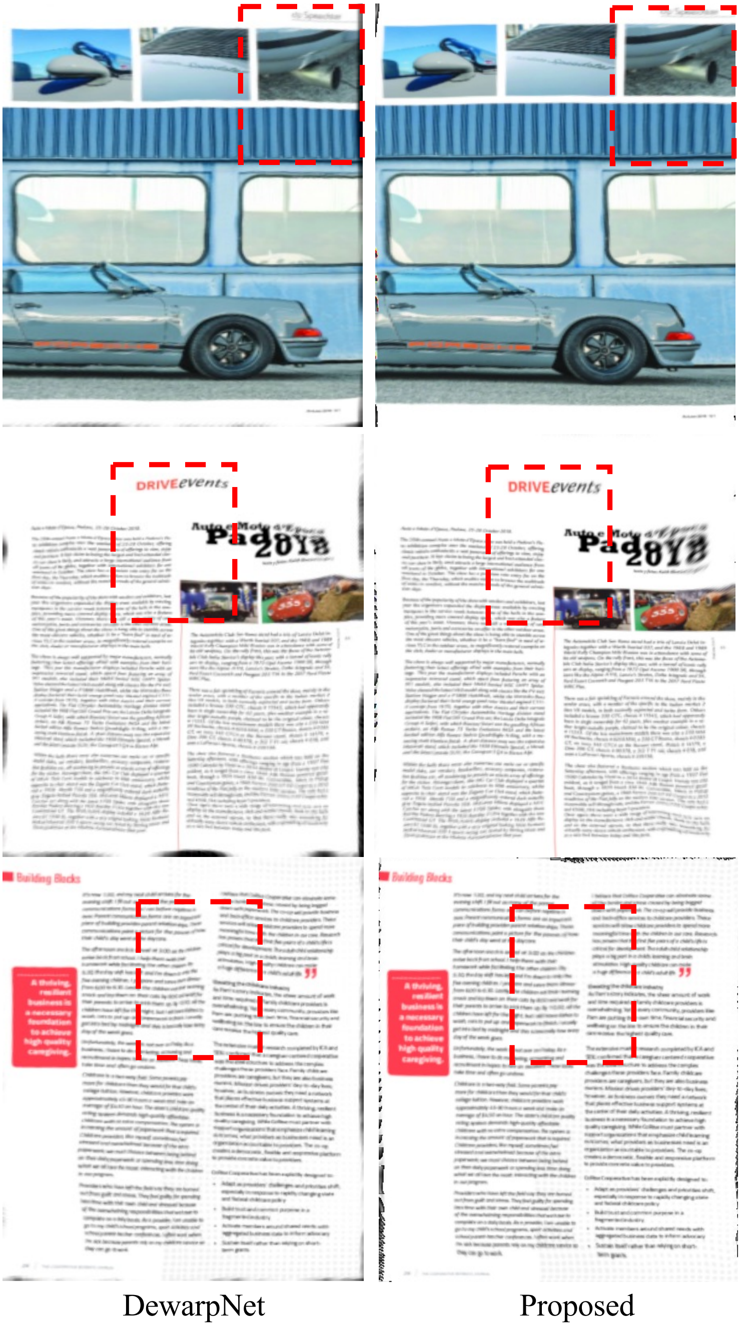


Fig. 8. Comparison of frontal view unwarping: left is DewarpNet and right is our approach. Our results are clearly better with straighter lines. Discriminative regions are highlighted with red dashed rectangles.

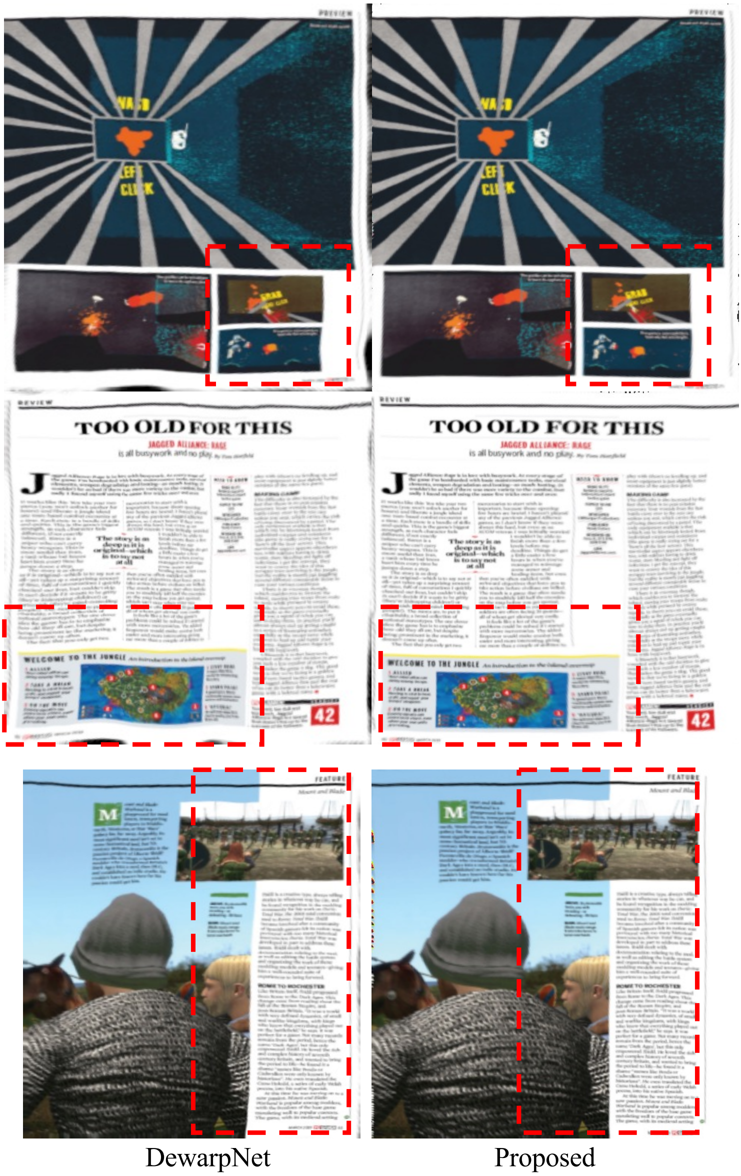
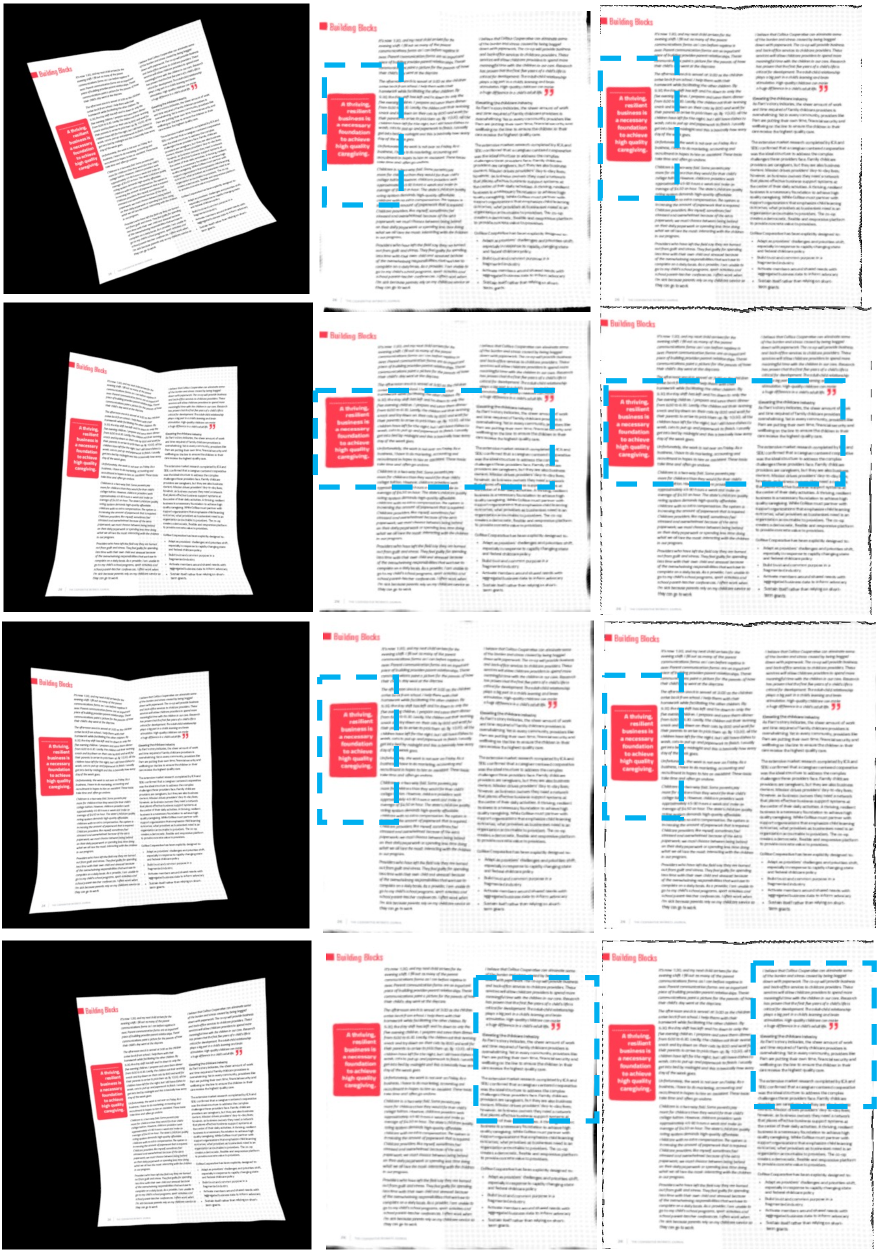


Fig. 9. Comparison of frontal view unwarping: left is DewarpNet and right is our approach. Our results are clearly better with straighter lines. Discriminative regions are highlighted with red dashed rectangles.

(a) (b) (c)

Fig. 10. 4 best results (sorted in ascending order from top to bottom according to LD score [lower better]) of (b) DewarpNet compared to (c) proposed unwarping for a specific scene. For all the views proposed unwarping shows better and consistent visual results than DewarpNet. (a) is the input. Blue dashed boxes denote the discriminative areas in the unwarped results.



(a) (b) (c)

Fig. 11. 4 best results (sorted in ascending order from top to bottom according to LD score [lower better]) of (b) DewarpNet compared to (c) proposed unwarping for a specific scene. In all the views proposed unwarping shows better and consistent visual results than DewarpNet. (a) is the input. Blue dashed boxes denote the discriminative areas in the unwarped results.

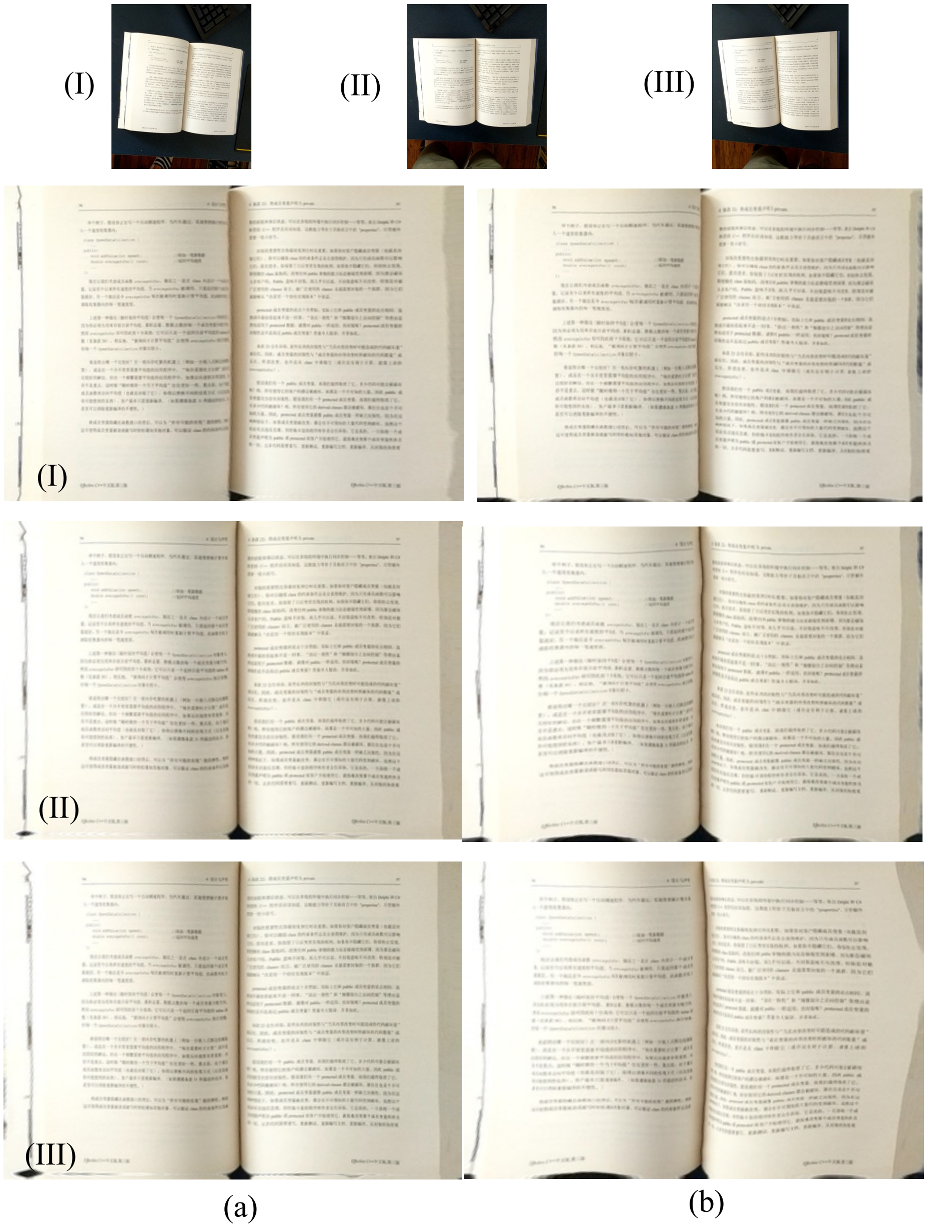


Fig. 12. Unwarping results on different views of a book. Top row shows the inputs. (a) Proposed, (b) DewarpNet. Our method generates good quality unwarping results with straighter text-lines.

Fig. 13. Unwarping results on different views of a receipt. Top row shows the inputs. (a) Proposed, (b) DewarpNet. Our method generates good quality unwarping results with straighter text-lines.

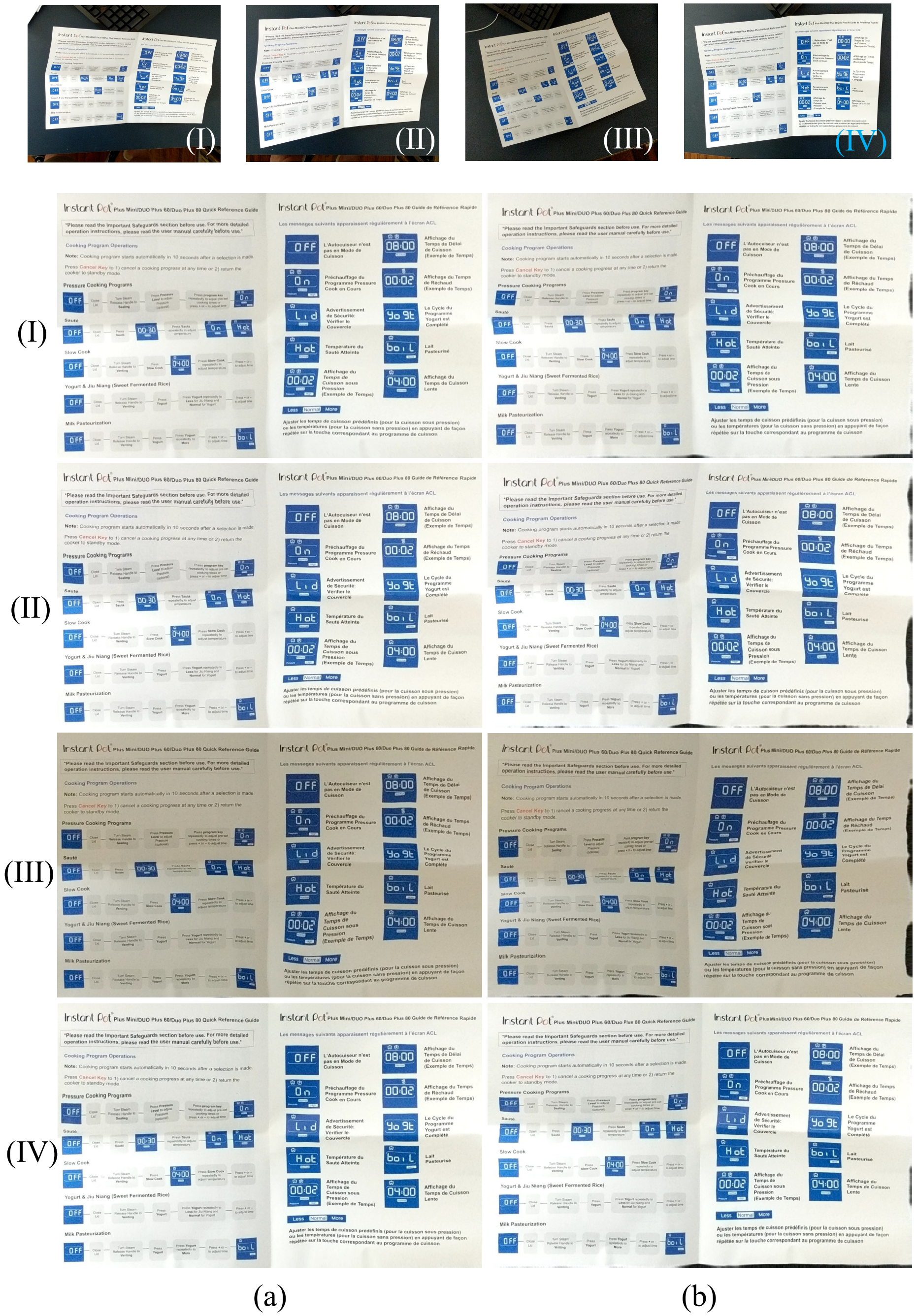


Fig. 14. Unwarping results on different views of a flyer. Top row shows the inputs. (a) Proposed, (b) DewarpNet. Our method generates good quality unwarping results with straighter text-lines.

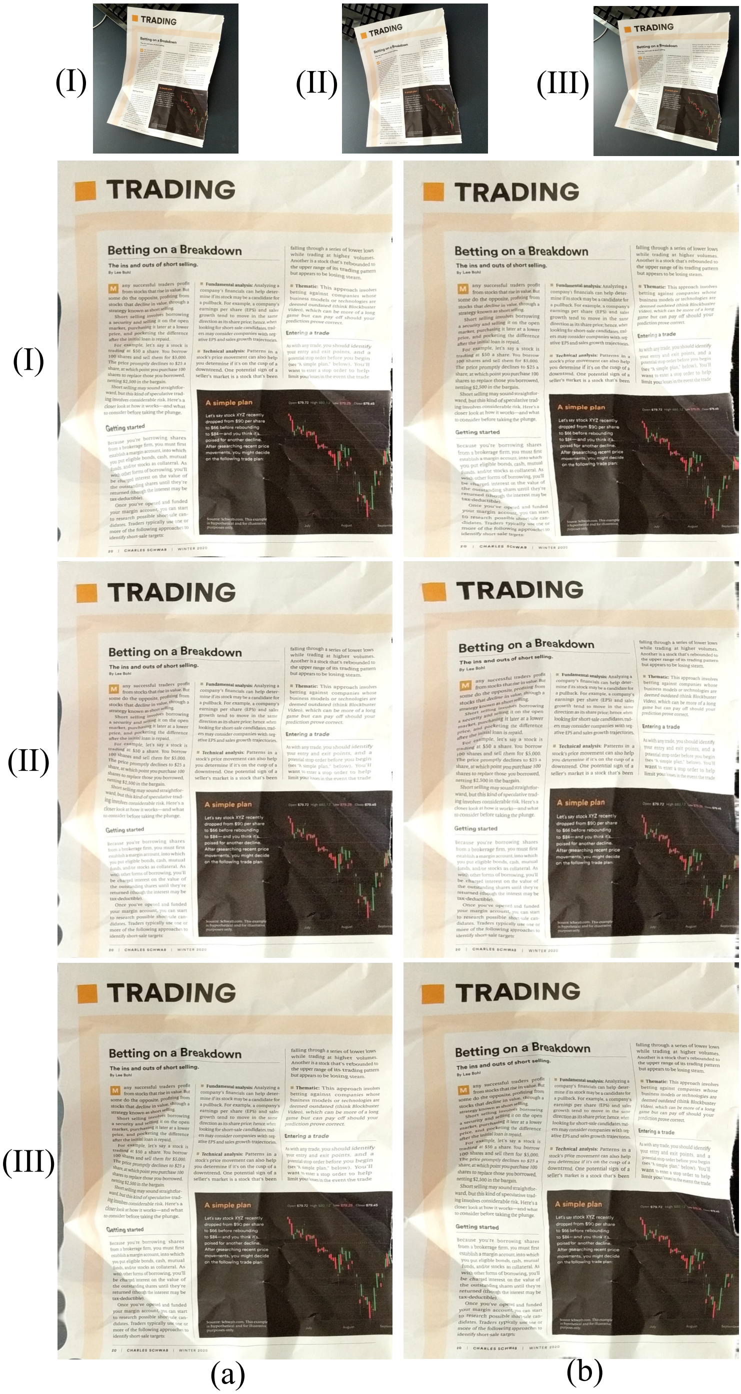
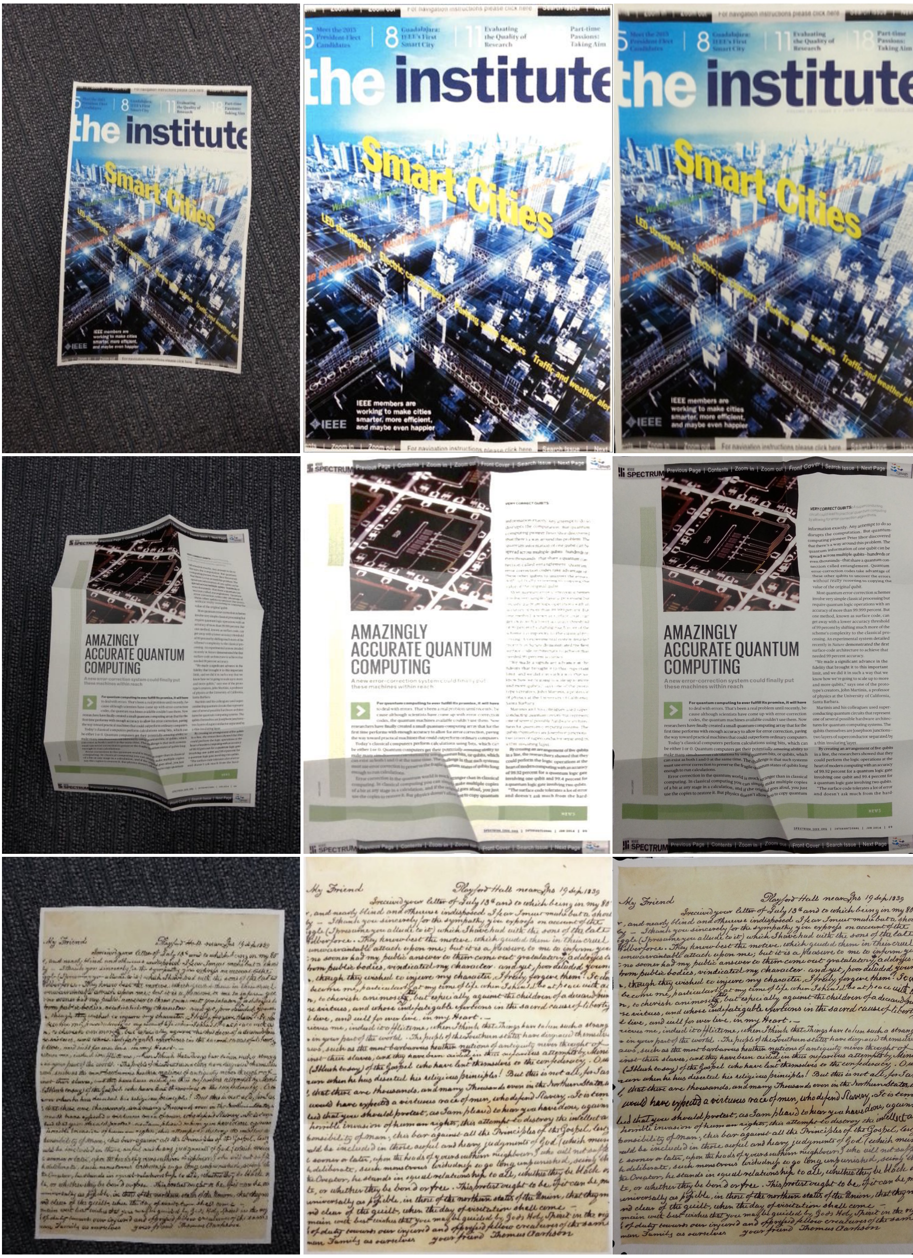


Fig. 15. Unwrapping results on different views of a magazine page. Top row shows the inputs. (a) Proposed, (b) DewarpNet. Our method generates good quality unwrapping results with straighter text-lines.



Input You et al. Proposed

Fig. 16. Comparison with [16]: We show competitive unwarping results compared to a prior multi-view unwarping approach. A quantitative comparison could not be performed because high-res/original unwarped results are not publicly available.

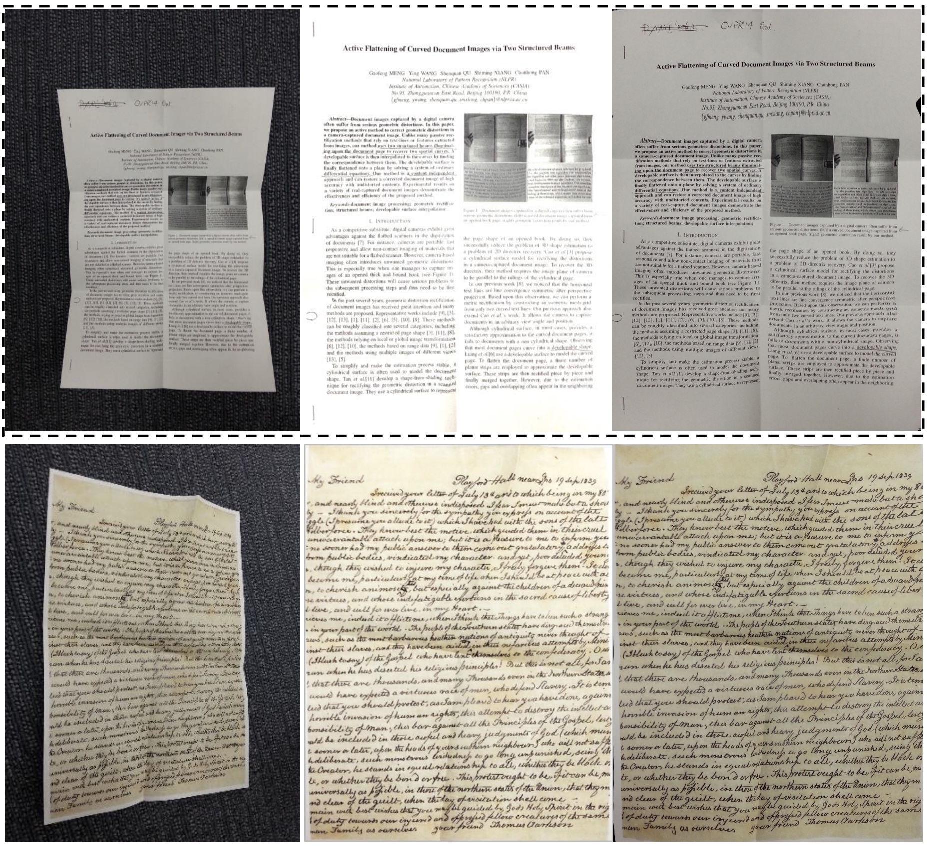
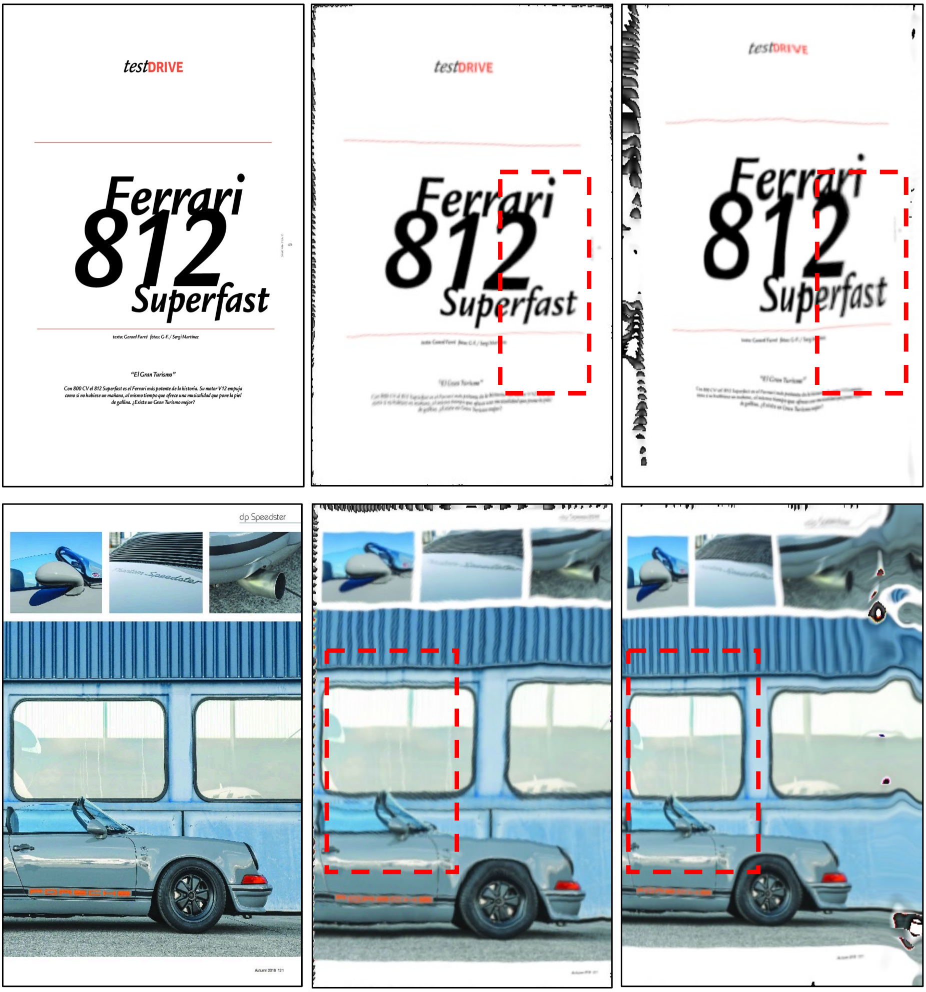


Fig. 17. Comparison with [16]: We show competitive unwarping results compared to a prior multi-view unwarping approach. A quantitative comparison could not be performed because high-res/original unwarped results are not publicly available. The example with the dashed outline shows a failure case of our method: 'Real 6'(see figure 21).



GT | With L_{uv} | Without L_{uv}

Fig. 18. Usefulness of L_{uv}: Examples trained **without** L_{uv} show undesired stretches and squeezes in the unwarped texture.

8 Details of weighting function used in L_z

We define L_z in Eq. 9 of the main submission:

$$L_z = \frac{1}{|P_{in}|} \sum_{p \in P_{in}} w_p (\hat{z}_p - \hat{z}'_p)^2 \tag{1}$$

where $P \in P_in$ are the pixels for which ray-surface intersection is found and $M_p = 1$. M_p, denote the pixel in the document mask M. M is a binary image, where $M_p = 1$ denotes the pixel p is within the document region. w_p is a pre-calculated per-pixel weight based on the document mask (M). $\hat{z}_p$ is the ray-surface intersection point obtained by sphere tracing, and $\hat{z}'_p$ is the ray-surface intersection point predicted by F_z. To derive the 2D texture map of a 3D surface, constraint optimization-based techniques use user-defined keypoints [14]. The keypoints allow to constrain the 2D to 3D mapping estimation. For documents, we can consider the set of boundary points as the keypoints. From the application perspective, it helps to accurately map the texture boundary to the learned surface boundary (see Fig. 20(d) vs. (e) vs. (f)). Therefore, we employ a weighting function, which assigns a higher weight to the 3D surface points at the boundary. To implement $W(p)$ we use a Euclidean distance transform [3] on the document mask M, a binary image. Each pixel p, in the distance transformed image, D encodes the distance to the nearest non-zero pixel. We first normalize and invert the distance transformed image:

$$\begin{aligned} D^{norm} &= \frac{D - \min(D)}{\max(D) - \min(D)} \\ D^{inv} &= 1 - D^{norm} \end{aligned}$$

Here max(.) and min(.) denote the maximum and minimum value of D_p over all the pixels. We assign the weights w_p as follows:

$$w_p = \begin{cases} 10.0, & \text{if } D_p^{inv} > 0.8 \\ 0.3, & \text{otherwise} \end{cases} \tag{2}$$

9 Training details of the UV prior network ($\hat{F}_{uv}$)

We use an 8 layer MLP with a hidden layer of 512 units to learn the 3D to UV mapping prior for document shapes. Each hidden layer has a sine [12] activation function. The final layer uses a HardTanh activation function. To train $\hat{F}_{uv}$ we utilize 10K UV mapped document meshes available in the Doc3D dataset. Each mesh is first registered with a $[-1, 1]$ uniform grid using a rigid transformation. Then the meshes are rendered in Blender [1] to obtain the projected geometry image (G) and the UV image (U). In G, each pixel p encodes the (X,Y,Z) coordinates. In U, p encodes the corresponding UV coordinates. During training, we randomly sample 10K pixels from each G as input to $\hat{F}_{uv}$ and use the corresponding pixels in U as the ground-truth. We optimize the L1 loss for 150 epochs between the predicted and the ground-truth UV coordinates using the Adam optimizer with an initial learning rate of 10^{-5}. The learning rate is halved every 50 epochs. Following NeRF [9], we use a high dimensional Fourier mapping ($\chi_k : \mathbb{R} \rightarrow \mathbb{R}^{2k}$) to learn high-frequency details in the shape and the UV space. We empirically set the number of Fourier bands, $k = 10$.

10 Initializing S and F_z

We can start optimizing S from the standard IDR initialization (SDF of a sphere). However, we notice that a better initialization can significantly improve the training time as well as the quality of the shape reconstruction. For object-specific applications like document unwarping, we found that initializing S with a similar object can significantly reduce the training time and converge in half the number of iterations (from 400K to 200K). Furthermore, we also found that initializing F_z to produce a planar point cloud can further reduce our training convergence time to ~ 6 hours (80K-100K iterations). To this end, we pre-train F_z to produce a plane.

Pre-training of F_z. To initialize F_z such that it produces a planar point cloud, we pre-train it by inputting points sampled from the UV space and predict the point cloud with $Z = 0$. We employ Chamfer distance as a loss function between ground truth and predicted 3D points. The ground truth points are sampled from a plane. Additionally, we also apply the conformality constraints (defined in section 3.2 of the main submission) for this pre-training step. The predicted plane is bounded in $[-0.5, 0.5]$ in our implementation. This training step is quite straightforward and converges in a few epochs.

11 Unwarping and texture editing details

To unwarp an input image, we determine a pixel at $p = (x, y)$ in the input image should be projected to (u, v) in the unwarped image. Here the unwarped image refers to the texture space. The coordinates (u, v) and p are associated by $F_{\mathbf{z}}$ and τ: For a (u, v) coordinate, its corresponding point in 3D is obtained by $\hat{z}'_p = F_{\mathbf{z}}(u, v)$. Given the camera parameter τ, $\hat{z}'_p$ is projected to p in the input image. Thus, we can find its corresponding pixel in the input image for each pixel in the unwarped image, which is all we need for unwarping. More specifically, we use standard image projection and bilinear sampling [7] to implement the unwarping step (see Fig. 19). The unwarping process can be realized as a grid sampling step from the warped document image to a 2D rectangular uniform grid. We can perform this sampling operation with a grid $G \in \mathbb{R}^{(H \times W \times 2)}$ and a bi-linear sampler. Here H and W denote the height and the width of the grid. Each location in G encodes a pixel coordinate $\hat{p}$ of the input image.

At test time we sample $F_{\mathbf{z}}$ in a uniform grid and project using the known camera pose (τ) to obtain the pixel coordinates. More specifically, sampling $F_{\mathbf{z}}$ in a uniform grid $\in [0, 1]$ yields a uniform 2D grid $R_z \in \mathbb{R}^{(H \times W \times 3)}$. Each (u, v) in R_z encodes a 3D coordinate of the document surface. The R_z representation of the 3D shape is analogous to geometry images [6]. We obtain G from R_z with a standard projection:

$$\hat{p} = K\ [R|T]\ \bar{\mathbf{z}} \tag{3}$$

Here, $\bar{\mathbf{z}}$ is the homogeneous coordinate representation of $\mathbf{z}$. $K \in \mathbb{R}^{3\times 3}$, $[R|T] \in \mathbb{R}^{4\times 4}$, denote the intrinsic and extrinsic parameters of the camera.

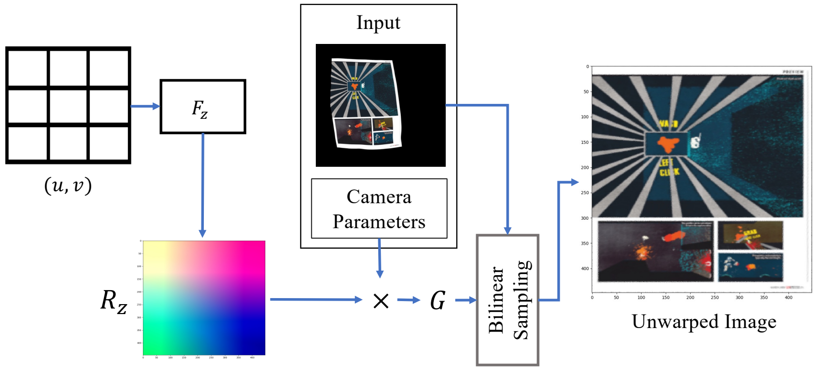


Fig. 19. Unwarping steps at test time: R_z denotes the flattened geometry in the texture space. Using the camera projection matrix for each view, we can obtain the unwarping grid G. $\times$ denotes matrix multiplication. G can be used to sample [7] the input image to get the unwarped image.

For the texture editing task, we first unwarp the image, then edit the texture, and finally warp each edited pixel p back to the original position using the predicted texture coordinates (t_p). We can utilize the same bilinear sampling operation as the unwarping step.

12 Pre-processing details for the real scenes

To train our proposed approach on the real scenes, we first obtain the camera poses using COLMAP [11]. Each scene in the real data has 5-25 views. We pre-process the camera poses to a spherical domain following [8]. Since all the training meshes used to train $\hat{F}_{uv}$ are aligned with a $[-1, 1]$ uniform grid, we apply a fixed pre-computed rigid-transformation on the estimated 3D shape during the joint training of S, F_{uv}, and F_z. Specifically, we use a $6D$ rigid transformation, with two parameters for rotation (axis-angle representation), three for translation, and one for scale. We first train a vanilla IDR [15] for 10K iterations. Then we obtain a 3D point cloud representation of the surface by sphere-tracing the IDR estimated SDF. Each point in the point cloud is a ray-surface intersection point. Note that we do not need a very accurate geometry at this step. Therefore it is not required to optimize the SDF until convergence. Now we obtain the desired rigid transformation by optimizing the Chamfer distance between the obtained surface point cloud and 10K points sampled from a 2D uniform regular grid $\in [-1, 1]$. We use SGD [13] with a learning rate of 0.001 and momentum 0.9 and optimize for 10K iterations. Later, At every iteration during the joint training, we apply the estimated rigid transformation on the sphere traced surface points $(\hat{z}_p)$ and use the transformed points as an input to the F_{uv}.

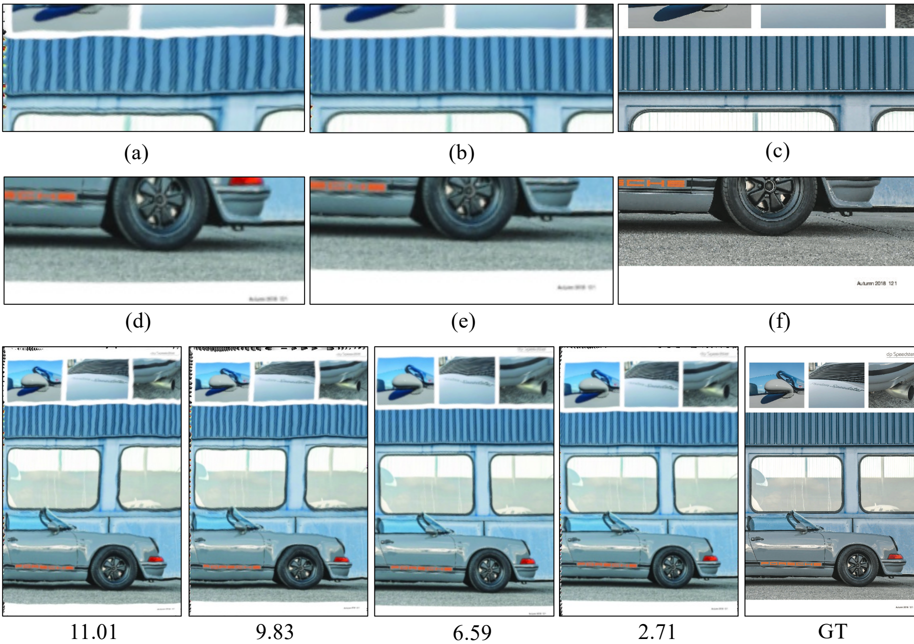


Fig. 20. Weighted L_z, and conformality effects. Top and middle row: (a) without conformality constraints, (b) with conformality constraints, (d) $w_p = 1$ in weighted L_z, (e) weighted L_z with w_p calculated using Eq. 2, (c,f) ground-truth; bottom-row (left-to-right): without conformality constraints and weighted L_z; only with weighted L_z; only with conformality constraints; with conformality constraints and weighted L_z; ground-truth scan. Numbers in bottom denote the respective LD values.

13 Detailed ablation figure

We show a more detailed example of Fig. 8 of the main submission in Fig. 20, with zoomed-in regions to demonstrate the effect of the different components of L_T (Eq. 10 in the main paper).

14 Limitations

In the following, we discuss few potential limitations of our method:

- **3D reconstruction:** The main limitation of our method follows from IDR [15]. Inadequate number of images of a scene with large texture-less regions lead to inferior 3D reconstruction which affects our unwarping result (see section 15).
- **Training time:** The current approach takes ∼6 hours to train a model and separate models must be trained for every scene which makes it unsuitable for real time applications. Runtime improvement will be addressed as a future work (see section 16).

– **Need for masks:** We assume masks are available for every image. Although masks are currently provided as manual inputs, we believe it's fairly straightforward to train a foreground-background segmentation models to automate the task.

15 Example of a failure case

Our method might fail due to imperfect 3D reconstruction. We show one such case for a scene from [16]. Mainly, there are two reasons for failure cases: first, fewer views (only 5), and second, insufficient textured documents. IDR has insufficient information to reconstruct the 3D shape. As a result of the poor 3D shape, our texture parameterization network produces an inferior unwarping result. For illustration, we show the reconstructed 3D shape, warped texture, and unwarped texture in Fig. 21.

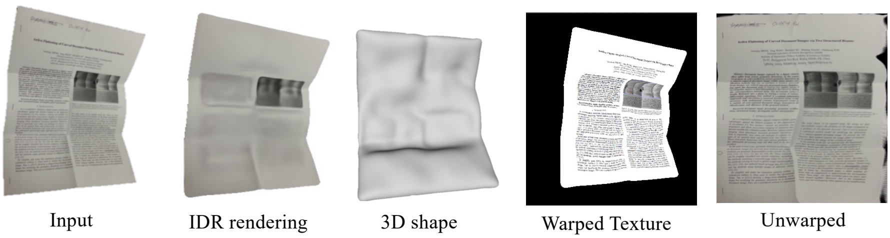


Fig. 21. Shows a failure case of our method due to inferior 3D reconstruction. This happened due to fewer available views for the scene and insufficient texture.

16 Training Time

Our proposed method for a scene can be trained in approximately 6 hours for 448×448 resolution images using a single Titan Xp GPU. The current training time per scene is high compared to DewarpNet's inference time which makes it unsuitable for real-time applications. However, we would like to note that in the current implementation sphere-tracing takes almost 50-60% of the running time. With a faster version of the sphere-tracing we can readily achieve a faster framework. Moreover, neural rendering is an active research field and there are multiple other works that are focusing on improving the speed and generalization abilities [5,2]. Therefore, a faster training can be achieved following any newer or faster alternatives of IDR.

References

1. Blender - a 3D modelling and rendering package

2. Bergman, A.W., Kellnhofer, P., Wetzstein, G.: Fast training of neural lumigraph representations using meta learning (2021)
3. Borgefors, G.: Distance transformations in digital images. Computer vision, graphics, and image processing **34**(3), 344–371 (1986)
4. Das, S., Ma, K., Shu, Z., Samaras, D., Shilkrot, R.: DewarpNet: Single-image document unwarping with stacked 3D and 2D regression networks. In: Proceedings of the International Conference on Computer Vision (2019)
5. Garbin, S.J., Kowalski, M., Johnson, M., Shotton, J., Valentin, J.: Fastnerf: High-fidelity neural rendering at 200fps (2021)
6. Gu, X., Gortler, S.J., Hoppe, H.: Geometry images. ACM Transactions on Graphics (TOG) **21**(3), 355–361 (2002)
7. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks. In: Advances in Neural Information Processing Systems (2015)
8. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) (2019)
9. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing scenes as neural radiance fields for view synthesis. In: Proceedings of the European Conference on Computer Vision (2020)
10. Ramon, E., Triginer, G., Escur, J., Pumarola, A., Garcia, J., Giro-i Nieto, X., Moreno-Noguer, F.: H3d-net: Few-shot high-fidelity 3d head reconstruction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5620–5629 (2021)
11. Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: European Conference on Computer Vision (ECCV) (2016)
12. Sitzmann, V., Martel, J.N., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions. In: arXiv (2020)
13. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International conference on machine learning. pp. 1139–1147. PMLR (2013)
14. Tzur, Y., Tal, A.: FlexiStickers: Photogrammetric texture mapping using casual images. In: Proceedings of the ACM SIGGRAPH Conference on Computer Graphics. Association for Computing Machinery (2009)
15. Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Ronen, B., Lipman, Y.: Multiview neural surface reconstruction by disentangling geometry and appearance. Advances in Neural Information Processing Systems **33** (2020)
16. You, S., Matsushita, Y., Sinha, S., Bou, Y., Ikeuchi, K.: Multiview Rectification of Folded Documents. IEEE Transactions on Pattern Analysis and Machine Intelligence (2017)