6. Appendix

6.1. CUDA Details

We have initially trained and tested our implementation of *Unsup3D* with CUDA 11.3 on a GPU instance of GeForce RTX 3090, which is compatible with CUDA versions of 11.0+. However, as shown in Fig. 8 the training processes of our implementation on both BFM and CelebA datasets have resulted in critical gradient explosions and in consequence failed to properly learn the decomposition of the input image to *depth, albedo, light* and *viewpoint*. Fig. 7 shows the problematic decomposition results from our model trained with such defects in the training process.

Training on CUDA 11.3. The author's original implementation[8] was only tested with CUDA 9.2, PyTorch 1.2.0 on CentOS 7. When we trained their code on CUDA 11.3 it suffered from sudden spikes in the losses during training (Fig. 8), and the decomposition results were also degenerated as shown in Fig. 7. Similar issues regarding the training of *Unsup3D* on newer versions of CUDA were also reported in official implementation repository[9][10]. The reports claim that the training loss does not converge when training the author's implementation on CUDA 11+ and the performance of the decomposition was degraded as our initial implementation did.

Applying gradient clipping. In order to prevent the training process of our implement from falling into such degenerations as in Fig. 7, we have introduced gradient clipping into our training code which *clips* the loss gradients when their absolute values exceed 5, thus blocking sudden loss explosions from occurring. Moreover, we also added the gradient clipping into the training code of the author's code and achieved successful training results with the author's code on CUDA 11.3. Fig. 8 shows that applying gradient clipping has led to more stable learning for both author's and ours compared to the original code without gradient clipping. As shown in Fig. 7, the introduction of gradient clipping has led the neural networks in *photo-geometric autoencoding* to properly predict the visual elements from input images.

Ablation on CUDA versions. As shown in Sec. 6.1, we compared the SIDE, MAD errors of *Unsup3D* on different versions of CUDA (9.2, 10.0 and 11.3) in order to confirm that the gradient explosion during training correlates with the CUDA versions. The results imply that training on CUDA 11.3 could have caused such training issues, as both SIDE, MAD errors on CUDA 11.3 highly exceed that of CUDA 9.2 and 10.0. We suppose that different CUDA versions can possess different priorities of operations, thus leading to race conditions or other unseen internal bugs in older versions. However, further researches would be required to understand the exact causes of such unexpected behaviors of different versions of CUDA.

[8] https://github.com/elliottwu/unsup3d
[9] https://github.com/elliottwu/unsup3d/issues/12
[10] https://github.com/elliottwu/unsup3d/issues/15

		SIDE ($\times 10^{-2}$)	MAD ($deg.$)
CUDA 9.2	Ours	-	-
	Author's	0.7792 ± 0.1387	16.11 ± 1.48
CUDA 10.0	Ours	1.0447 ± 0.2402	19.05 ± 2.19
	Author's	0.7796 ± 0.1372	15.93 ± 1.47
CUDA 11.3	Ours	4.4603 ± 0.7972	69.14 ± 3.81
	Author's	2.4585 ± 0.3247	35.35 ± 3.82

Table 3. **Ablation on CUDA versions (trained without gradient clipping).**

6.2. Network Details

For the prediction of albedo a, confidence map σ, and depth map d, *Unsup3D* uses auto-encoder composed of convolution layers and transpose convolution layers with group normalization. *Unsup3D* doesn't use residual connection because the albedo a, confidence map σ, and depth map d are oriented to canonical image J, not to the input image I. Encoders with convolution layers and group normalization is used to predict the light information l, k_s, k_d and the viewpoint w. All encoders in *Unsup3D* use Leaky ReLU instead of ReLU. You can check details in *unsup3d/modules.py*.

6.3. Rendering Details

Applying NMR instead of camera projection matrix We need to rotate and translate the viewpoint of canonical image to reconstruct original image. However, the depth map d is distorted since it is projected by camera projection matrix K. It's why we multiply inverse of projection matrix K^{-1} to calibrate this distortion. After rotating and translating the viewpoint, we can reconstruct original image by multiplying moved 3D points P with camera projection matrix K. Here, instead of simply multiplying the points, *Unsup3D* applies Neural Mesh Renderer. As it projects the mesh, not point cloud, the projected output doesn't have discontinuous empty pixels on foreground.
The projected mesh is generated simply. First 3D points P are used as vertices. Then we tessellated 2x2 adjacent points (3D points P has 1-on-1 matching 2D pixels p_{uv}, so we can define adjacent points in 3D as neighboring pixels in 2D) in two triangles which will be used as faces of mesh.

Grid Sampling Instead of directly rotating and translating canonical image J with above mentioned method, *Unsup3D* uses grid sampling to vary a viewpoint. The sampling matrix which maps the reconstructed image pixel $\hat{I_{uv}}$ to canonical image pixel J_{uv} is obtained by rotating and translating the depth map d. Original paper mentions that this methods has faster backpropagation and more stable gradient. [13]

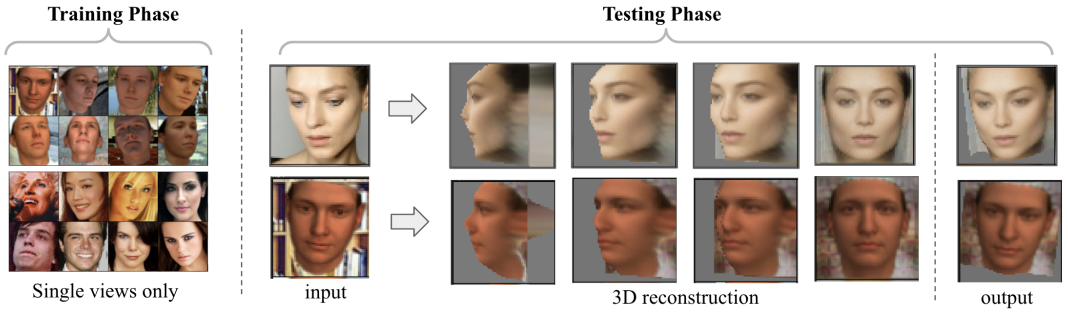

Figure 3. **Training, testing phases of *Unsup3D*** (Left) *Unsup3D* is trained on datasets composed of only single-view images, with no prior information of the 3D structures. (Right) From an input image, *Unsup3D* can predict the underlying 3D structure of the object and derive a reconstruction of the input.

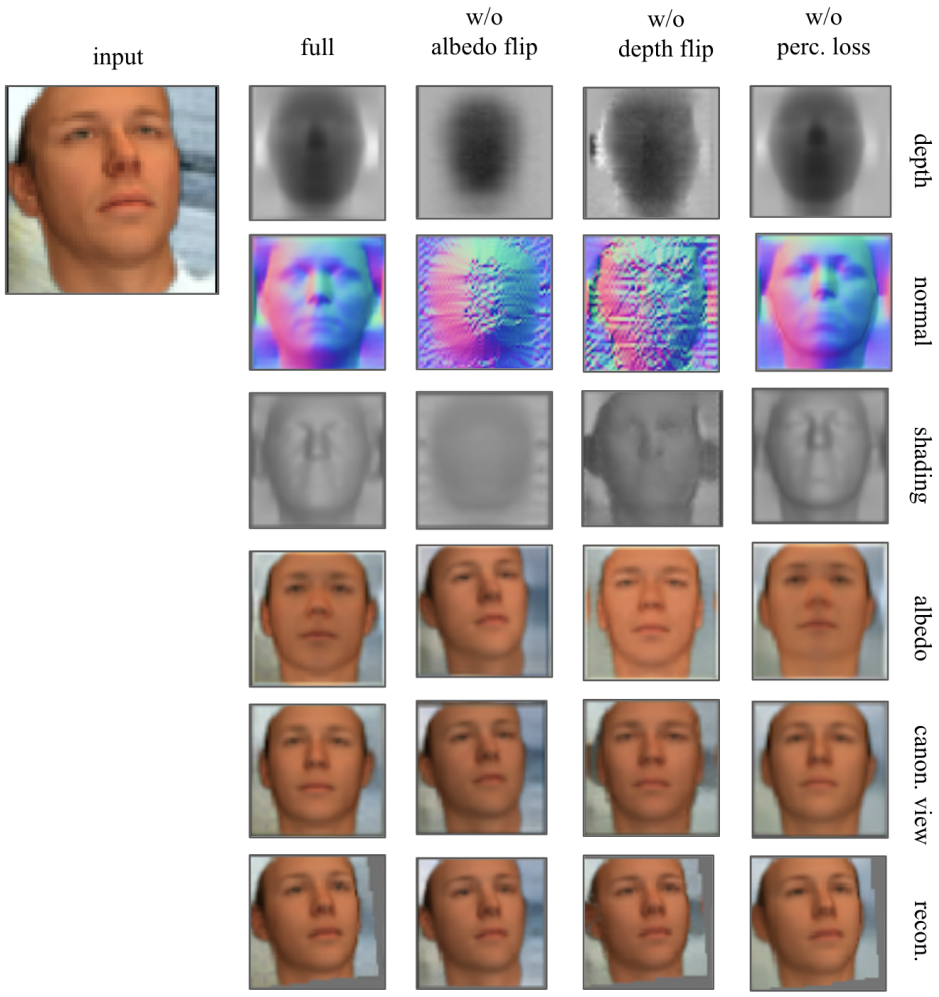

Figure 4. **Qualitative comparisons of the ablated models.**

Figure 5. **Reconstruction of human faces from CelebA dataset.**

Figure 6. **Reconstruction of human faces from BFM synthetic dataset.** The reconstruction is rendered from multiple viewpoints.

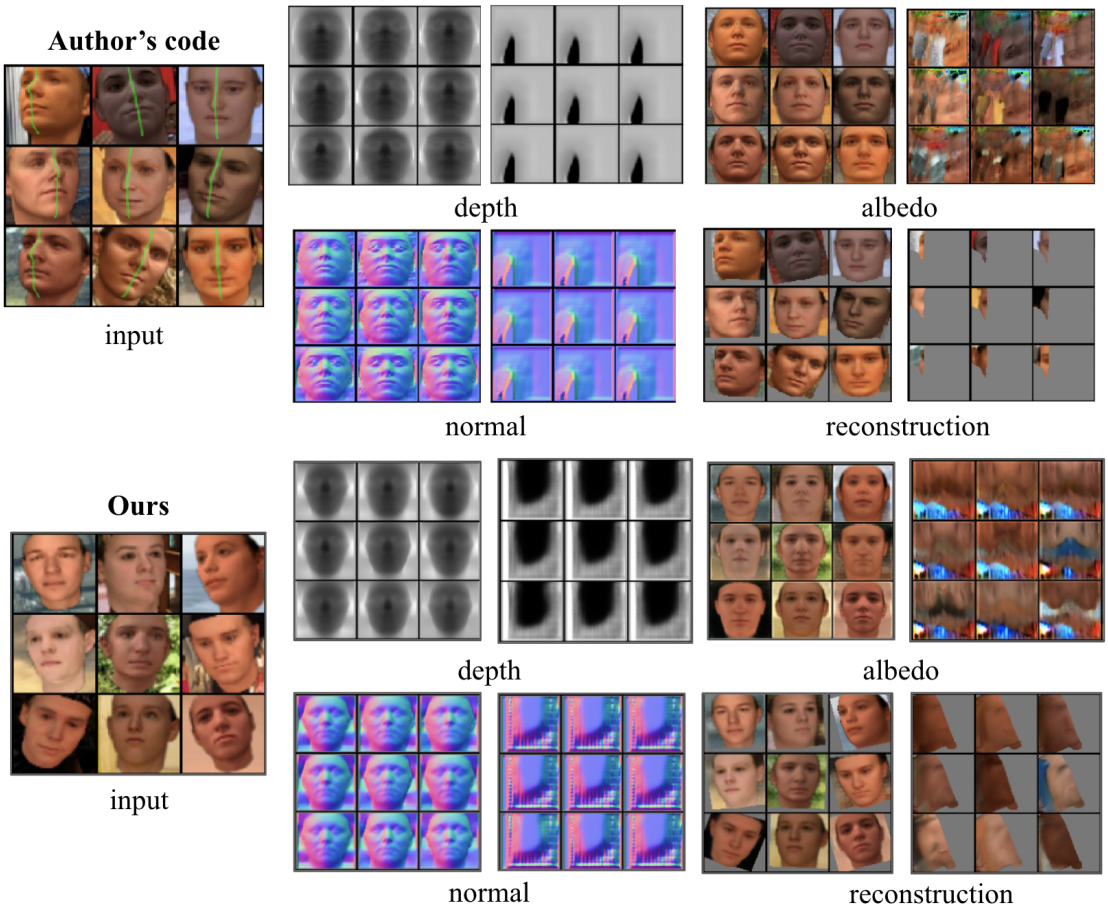

Figure 7. **Problematic training outputs on Cuda 11.3. (Top) Author's code.** (Left) Trained with gradient clipping (Right) Original code (No gradient clipping). **(Bottom) Ours.** (Left) Trained with gradient clipping (Right) Trained without gradient clipping.

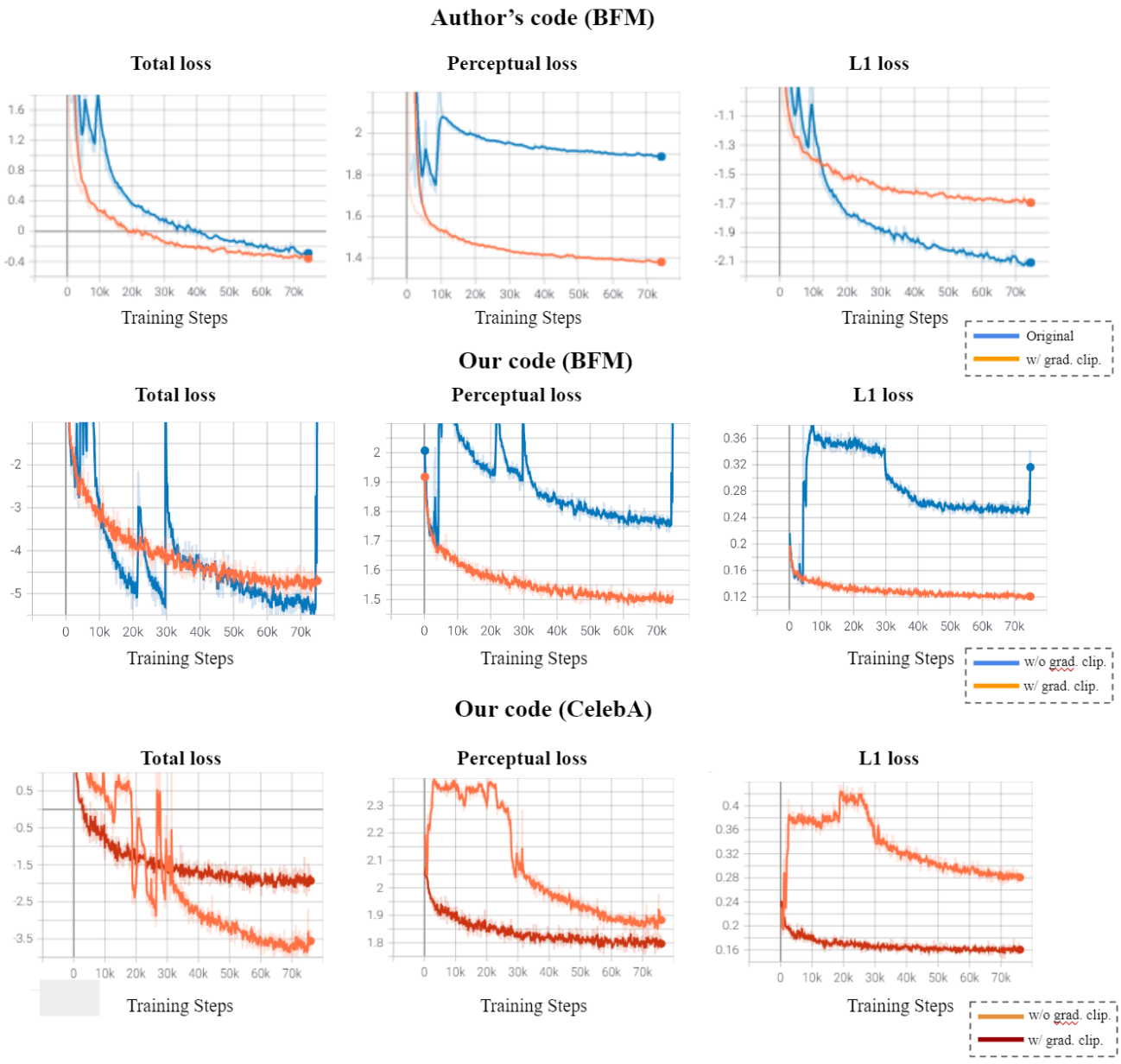

Figure 8. **Comparison of training losses with and without gradient clipping on CUDA 11.3.**