CORE-MPI: Consistency Object Removal with Embedding MultiPlane Image

Supplementary Material

A. Introduction

In the supplementary material, we provide the following details:

- The limitations of Pre-inpaint, including the inpainting result.
- Object removal in novel view synthesis based on single image MPI.
- Experiments for embedding image capacity.
- Visualization of merging two inpainting branch results.
- Detailed information about training loss.
- Detailed overall network architecture.

Moreover, we provide additional results as video samples.

B. Limitations of Pre-inpaint

This section describes the limitations of Pre-inpaint, including the inpainted stereo images. Figure 1 shows stereo images before and after object removal, together with the disparity maps of MPI generated from both images. Masked areas with objects in the original image are removed and restored by the inpainting model [3]. In the first example, the disparity map shows that differences in the region where the chair is removed lead to significant distortions in the MPI. Furthermore, the second example illustrates a failure in MPI generation due to the discrepancies in the inpainted regions. Since pre-inpaint requires the same number of object masks as the input images and shows distortions in the disparity map, we consider pre-inpaint to be suboptimal.

C. Single Image Novel View Synthesis

An alternative scenario for object removal in MPI involves removing objects from a single image and generating MPI using a single image MPI generator. In this section, we compare the performance of CORE-MPI with approaches that use the inpainting model [3] for object removal followed by single image MPI models, MINE [2] and Ada-MPI [1]. For a fair comparison, we set the number of MPI layers to 32 and measure the quality of the rendered view compared to the target frame adjacent to the reference image from RealEstate10K. As reported in Table 1, the quality of CORE-MPI results is superior, while the other approaches that estimate depth from a single image produce results that differ from the target frame.

D. Embedding Image Capacity

We evaluate the quality of the embedding image and the rendered images without object removal to validate the embedding performance of CORE-MPI. Moreover, to evaluate the potential of the encoding capacity, we conduct experiments by varying the depth layer n in StereoMag [4]. As shown in Table 2, the quality of the embedding image degrades as n increases, with a significant degradation observed when n is set to 128. While the quality of the rendered view shows a slight improvement as n increase from 32 to 64, it significantly worsens at 128. Future advancements in steganography are expected to address a limitation.

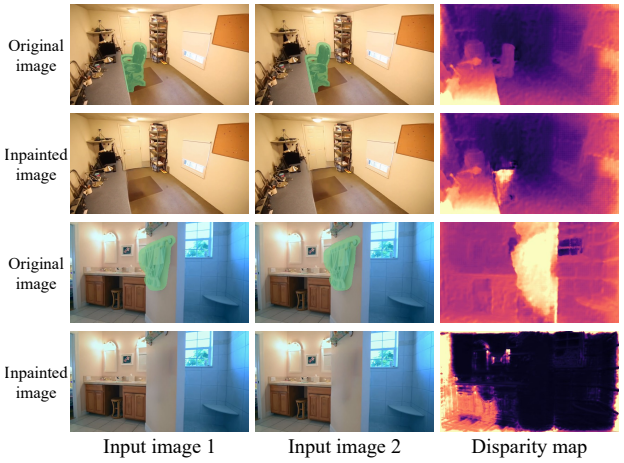

Figure 1. Pre-inpaint visualization: Inconsistencies in the inpainted region following object removal from the stereo image obstruct MPI generation, leading to distortions in the disparity map.

Model	PSNR↑	SSIM↑	LPIPS↓
MINE [2]	18.349	0.593	0.3565
Ada-MPI [1]	17.945	0.585	0.3464
CORE-MPI	**21.354**	**0.720**	**0.3294**

Table 1. Comparison of performance between single-image MPI generators, which remove object before MPI generation, and CORE-MPI.

	Embedding			Render		
n	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
32	44.214	0.998	0.002	36.146	0.974	0.026
64	43.040	0.997	0.005	37.772	0.987	0.016
128	35.489	0.995	0.044	29.183	0.910	0.120

Table 2. Qualitative results of embedding performance based on the number of depth layers.

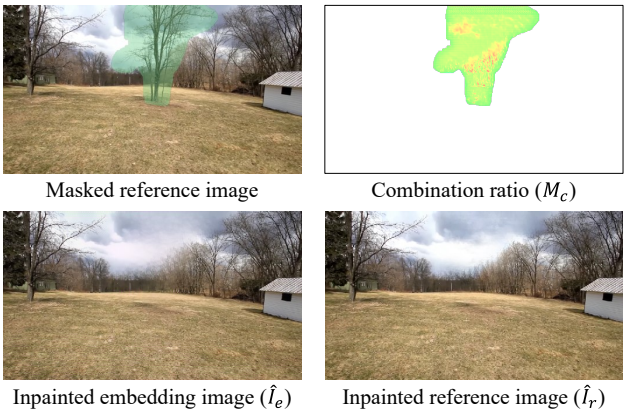

Figure 2. Intermediate steps visualization in merging inpainted images. M_c highlights the area where the filled embedding image is blurry, in contrast to the sharply restored reference image.

E. Visualization of Merging Process

Here we present a visualization of the intermediate steps involved in merging inpainted images. Figure 2 shows the inpainted embedding image $\hat{I}_e$ and the inpainted reference image $\hat{I}_r$ and the combination ratio M_c. Notably, M_c highlights the areas where the filled embedding image appears blurry compared to the sharply restored areas in the reference image. This visual representation illustrates the importance of the fusion module to detail in the merging process.

F. Detailed Information on Training Loss

This section describes the details of the loss functions used in CORE-MPI. For the training of the restoration network, the novel view loss uses randomly sampled values within the range of -0.05 to 0.05 for offsets in the x, y, and z dimensions on the RealEstate10K dataset. For the UCSD dataset, these values are sampled from a broader range of -0.5 to 0.5. The weights assigned to the various loss components used in CORE-MPI are shown in Table 3.

Weight Value	RealEstate10K	UCSD
λ_{er}	10	8
λ_{ep}	12	8
λ_{fr}	10	10
λ_{fp}	30	30
λ_{Ad}	3	3
λ_{pse}	2	2
λ_{gp}	0.001	0.001
λ_{fm}	100	100
λ_{c}	300	300
λ_{α}	30	30
λ_{nr}	100	100
λ_{np}	20	30
λ_{dr}	50	80
λ_{dsm}	1	1

Table 3. Hyperparameters of training loss.

G. Detailed Architecture

We provide the architecture of our proposed network within CORE-MPI. The detailed structures of the embedding network, the fusion module, and the restoration network are presented in Table 4, Table 5, and Table 6, respectively. In these tables, cat denotes the concatenation operation, while $+$ denotes element-wise summation.

References

[1] Yuxuan Han, Ruicheng Wang, and Jiaolong Yang. Single-view view synthesis in the wild with learned adaptive multi-plane images. In *ACM SIGGRAPH*, 2022. 1

[2] Jiaxin Li, Zijian Feng, Qi She, Henghui Ding, Changhu Wang, and Gim Hee Lee. Mine: Towards continuous depth mpi with nerf for novel view synthesis. In *ICCV*, pages 12578–12588, 2021. 1

[3] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. In *WACV*, pages 2149–2159, 2022. 1

[4] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: learning view synthesis using multiplane images. *ACM TOG*, 37(4):1–12, 2018. 1

Block	Input	Layer	Channels	Stride
Feature extractor				
RGB extractor	C	Group conv ReLU Group conv	96 → 96 96 → 32	1×1 1×1
Alpha extractor	α	Conv ReLU Conv	32 → 32 32 → 32	1×1 1×1
Ref conv	I_r	Conv ReLU Conv	3 → 16 16 → 3	1×1 1×1
UNet				
Init conv	$cat(f_{MPI}, f_r)$	Conv	67 → 64	1×1
ResBlock1	Init conv	ReLU Conv ReLU Conv	 64 → 64 64 → 64	 1×1 1×1
Down1	ResBlock1	ReLU Conv ReLU Conv	 64 → 128 128 → 128	 2×2 1×1
Down2	Down1	ReLU Conv ReLU Conv	 128 → 256 256 → 256	 2×2 1×1
ResBlock2	Down2	ReLU Conv ReLU Conv	 256 → 256 256 → 256	 1×1 1×1
ResBlock3	ResBlock2	ReLU Conv ReLU Conv	 256 → 256 256 → 256	 1×1 1×1
ResBlock4	ResBlock3	ReLU Conv ReLU Conv	 256 → 256 256 → 256	 1×1 1×1
ResBlock5	ResBlock4	ReLU Conv ReLU Conv	 256 → 256 256 → 256	 1×1 1×1
UP1	ResBlock5	Upsample ReLU Conv ReLU Conv	 256 → 128 128 → 128	 1×1 1×1
UP2	UP1 + Down1	Upsample ReLU Conv ReLU Conv	 128 → 64 64 → 64	 1×1 1×1
ResBlock6	UP2 + ResBlock1	ReLU Conv ReLU Conv	 64 → 64 64 → 64	 1×1 1×1
ResBlock7	ResBlock6	ReLU Conv ReLU Conv	 64 → 64 64 → 64	 1×1 1×1
Final Conv	ResBlock7	ReLU Conv	 64 → 3	 1×1
Final Activation	Final Conv +I_r	Tanh		

Table 4. Embedding network architecture. Embedding network is composed of two downsampling blocks, four residual blocks, and two upsampling blocks. In addition, the reference image is incorporated directly through a skip connection to preserve detail. **Conv** refers to a 2D convolutional layer with a 3×3 kernel size. **Group conv** indicates a convolutional layer with groups equal to the number of MPI layers, and **ReLU** is an activation function that sets negative values to zero. Downsampling is performed using 2×2 stride convolutional layers, while upsampling involves increasing resolution by a factor of two using the nearest neighbor approach. C denotes the color channels of the MPI, α refers to the transparency channels of the MPI, and I_r represents the reference image. f_{MPI} and f_r are the feature maps extracted from MPI and the reference image, respectively, by feature extractor. **Tanh** scales the output to between -1 and 1.

Block	Input	Layer	Channels	Stride
Down1	$cat(\hat{I}_e, \hat{I}_r, M)$	Conv	$7 \rightarrow 32$	1×1
		ReLU		
		Conv	$32 \rightarrow 32$	1×1
		MaxPool		2×2
Down2	Down1	ReLU		
		Conv	$32 \rightarrow 64$	1×1
		ReLU		
		Conv	$64 \rightarrow 64$	1×1
		MaxPool		2×2
Mid	Down2	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 128$	1×1
UP1	Mid + Down2	ReLU		
		Deconv	$128 \rightarrow 64$	2×2
		Conv	$64 \rightarrow 64$	1×1
		ReLU		
		Conv	$64 \rightarrow 64$	1×1
UP2	UP1 + Down1	ReLU		
		Deconv	$64 \rightarrow 32$	2×2
		Conv	$32 \rightarrow 32$	1×1
		ReLU		
		Conv	$32 \rightarrow 32$	1×1
ConvBlock	UP2	ReLU		
		Conv	$32 \rightarrow 1$	1×1
		Sigmoid		

Table 5. Fusion module architecture. **Conv** refers to a convolutional layer with a 3×3 kernel size and **ReLU** is an activation function that sets negative values to zero. **Maxpool** is a 2D maxpooling layer with a 2×2 kernel size and **Deconv** indicates a 2D convtranspose layer for upsampling. $\hat{I}_e$ denotes the inpainted embedding image, $\hat{I}_r$ refers to the inpainted reference image and M is the object mask. **Sigmoid** represents the sigmoid activation function, which maps the output into a probabilistic range between 0 and 1.

Block	Input	Layer	Channels	Stride
ConvBlock	$\hat{I}_m$	Conv	$3 \rightarrow 64$	1×1
ResBlock1	ConvBlock	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock2	ResBlock1	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock3	ResBlock2	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock4	ResBlock3 + ConvBlock	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock5	ResBlock4	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock6	ResBlock5 + ResBlock3 + ConvBlock	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock7	ResBlock6	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ResBlock8	ResBlock7	ReLU		
		Conv	$64 \rightarrow 128$	1×1
		ReLU		
		Conv	$128 \rightarrow 64$	1×1
ConvBlock	ResBlock8 + ResBlock5 + ResBlock3 + ConvBlock	ReLU		
		Conv	$64 \rightarrow 256$	1×1
		Conv	$256 \rightarrow 128$	1×1
		Tanh		

Table 6. Restoration network architecture. Restoration network consists of eight residual blocks with the same number of channels, three convolution layers, and three skip connections are after the third, fifth and eighth residual blocks. **Conv** refers to a convolutional layer with a 3×3 kernel size and **ReLU** is an activation function that sets negative values to zero. $\hat{I}_m$ denotes the merged embedding image. **Tanh** scales the output to between -1 and 1.