MatryODShka: Real-time 6DoF Video View Synthesis using Multi-Sphere Images — Supplemental Document —

Benjamin Attal[1,2] [0000−0002−0132−5232], Selena Ling[1] [0000−0001−6458−4488], Aaron Gokaslan[1] [0000−0002−3575−2961], Christian Richardt[3] [0000−0001−6716−9845], and James Tompkin[1] [0000−0003−2218−2899]

[1]Brown University, USA [2]Carnegie Mellon University, USA [3]University of Bath, UK

These appendices contain additional results and comparisons (Section 1) as well as implementation details of our approach, including our used hardware and software (Section 2.1), our joint bilateral upsampling (Section 2.2), details of our architecture and hyperparameters (Section 2.3), and rendering pseudocode (Section 2.4).

1 Additional Results and Comparisons

We show additional results and comparisons in Figures 1 to 6, and in video form in our supplemental materials. This includes two additional MSI decompositions in Figures 1, 3 and 5, and view synthesis comparisons to the perspective double-plane-sweep baseline in Figures 2, 4 and 6. Our approach produces better novel views at larger synthesis baselines than the baseline method. We illustrate the limitations of a layered representation such as ours in Figure 7. Please see our supplemental video for additional results.

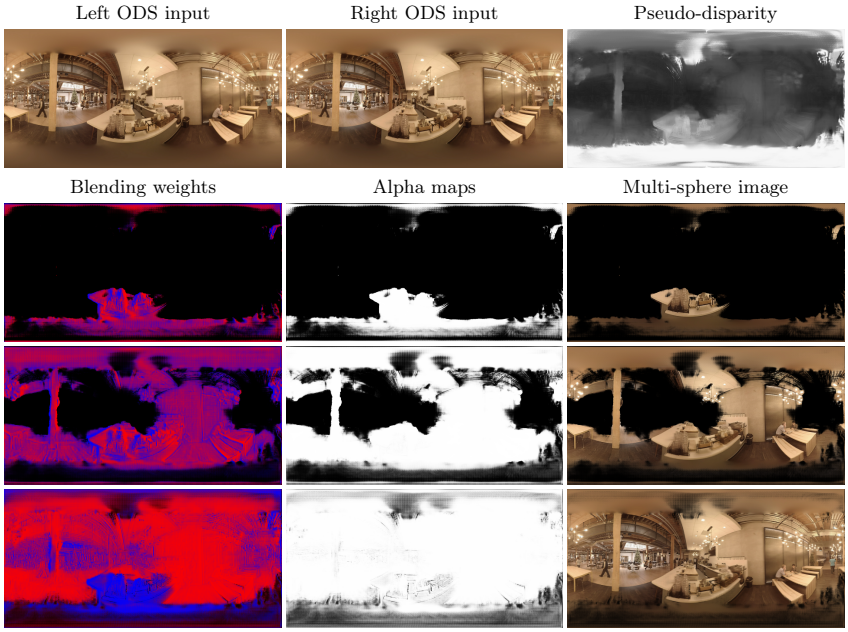

Fig. 1. Inferred MSI representation for the *Cafeteria* video [2]. Blending weights are red for left ODS and blue for right ODS. Alpha maps are black for transparent and white for opaque. Each row shows a single layer (out of 32) at the near, mid, and far extents of the scene; content exists across all layers to produce the final result.

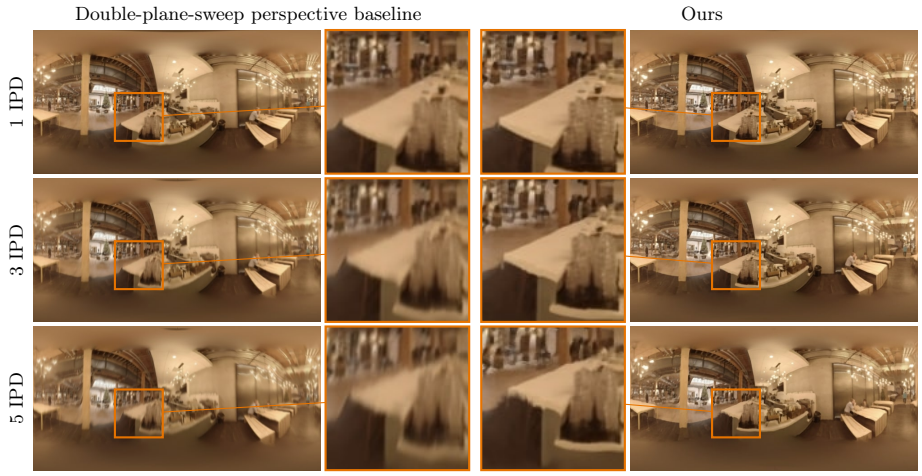

Fig. 2. *Cafeteria* video [2] results and comparison to the double-plane-sweep baseline on the left. Inference for low-resolution (640×320) MSI representation for comparison on spherical images (not our high-resolution real-time perspective VR view).

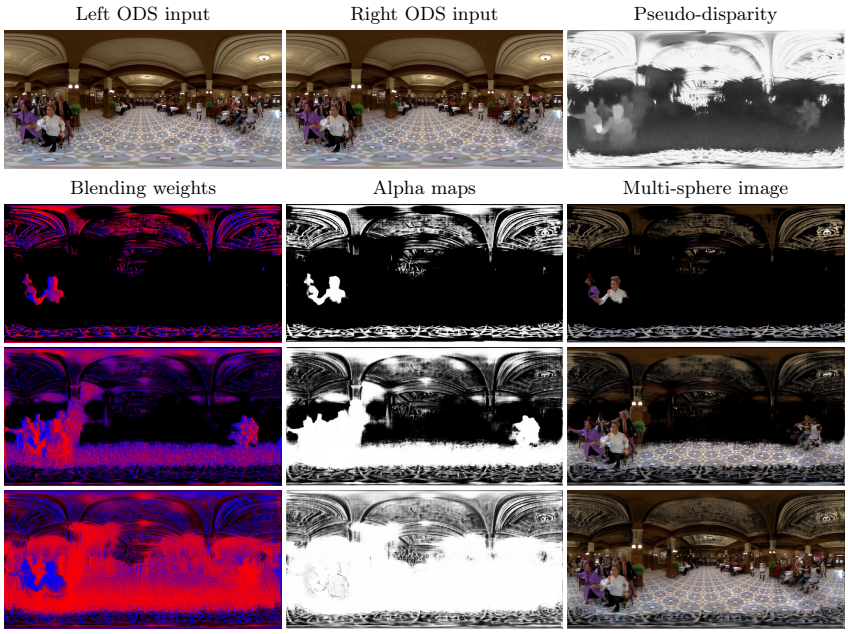

Fig. 3. Inferred MSI representation for the *MammaMia* video captured with a moving camera. Blending weights are red for left ODS and blue for right ODS. Alpha maps are black for transparent and white for opaque. Each row shows a single layer (out of 32) at the near, mid, and far extents of the scene; content exists across all layers to produce the final result.

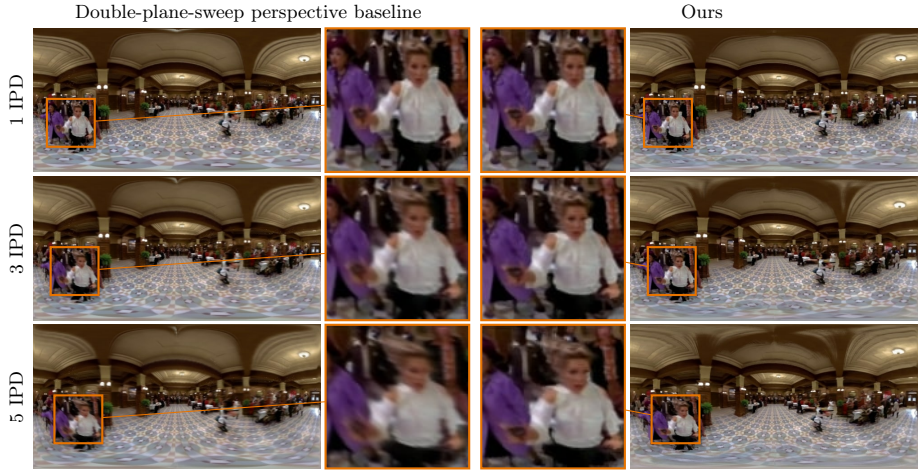

Fig. 4. *MammaMia* video results and comparison to the double-plane-sweep baseline on the left. Inference for low-resolution (640×320) MSI representation for comparison on spherical images (not our high-resolution real-time perspective VR view).

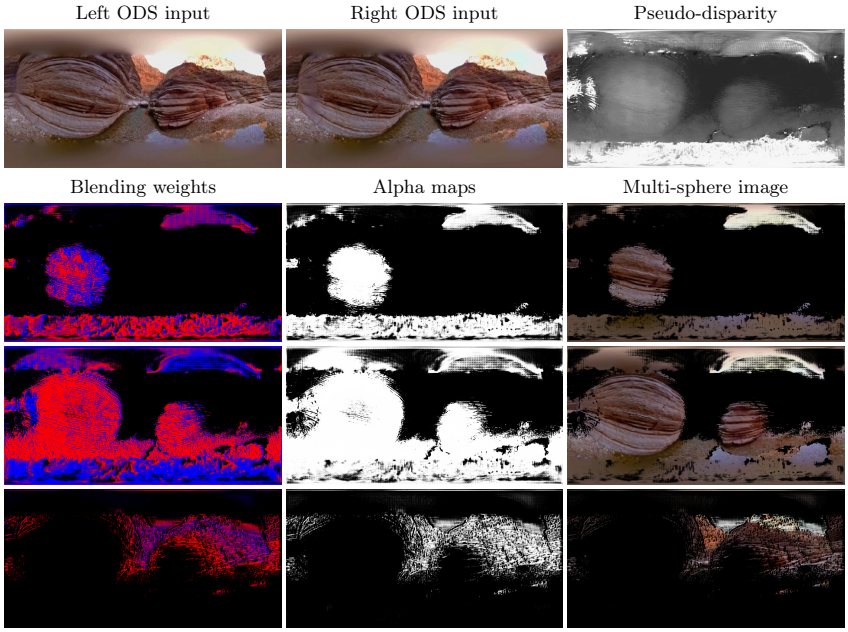

Fig. 5. Inferred MSI representation for the *GrandCanyon* video. Blending weights are red for left ODS and blue for right ODS. Alpha maps are black for transparent and white for opaque. Each row shows a single layer (out of 32) at the near, mid, and far extents of the scene; content exists across all layers to produce the final result.

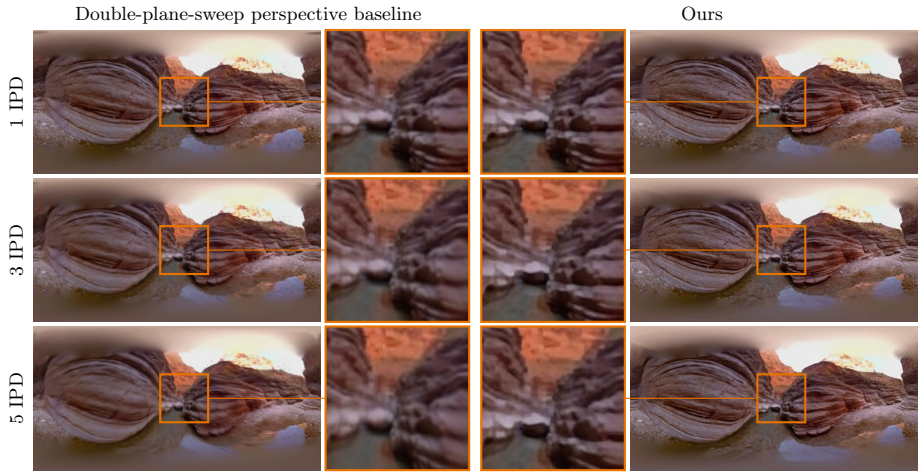

Fig. 6. *GrandCanyon* video results and comparison to the double-plane-sweep baseline on the left. Inference for low-resolution (640×320) MSI representation for comparison on spherical images (not our high-resolution real-time perspective VR view).

Fig. 7. *Limitation:* Using our system within a VR headset allows large motions away from the center of the MSI, exposing the layer structure of the representation (*bottom*).

2 Implementation Details

2.1 Hardware and Software

Simultaneously decoding high-resolution video (e.g., 4K×4K at 30 Hz), inferring MSIs, and rendering stereo video from MSIs into a VR headset requires significant compute. For the headset, we use an Oculus Rift S, which requires rendering at 80 Hz. We compare two current PC platforms: a current discrete-GPU laptop and a workstation. The laptop has an Intel(R) Core(TM) i7-8750H CPU with 16 GB RAM, and NVIDIA GeForce RTX 2080 with Max-Q Design. This provides 30 Hz MSI inference and 60+ Hz novel-view rendering (30+ Hz in VR). This rendering speed is sufficient for general desktop viewing, but not sufficient for fast head motion in VR. The workstation has an AMD Ryzen 2950X CPU with 64 GB RAM, and two NVIDIA GeForce 2080 Ti GPUs with RTX bridge to allow fast inter-GPU memory copy. This provides 30 Hz MSI inference and 250+ Hz novel-view rendering (125+ Hz in VR).

We train our model using TensorFlow via our TensorFlow-based reprojection renderer. For our inference engine within our Unity-based renderer, we use TensorRT for efficient GPU computation. We convert our model weights to 16-bit floating-point precision and load the model in TensorRT using ONNX. Further, we use CUDA for anti-aliased video downsampling to the network's expected input resolution, and for ODS sphere sweep volume creation. Finally, we implement ODS reprojection rendering from our MSIs, and our joint-bilateral upsampling, using Unity's CG shaders.

2.2 Joint-Bilateral Upsampling Effect

Our network architecture uses learned upsampling within its U-Net via transpose convolution layers, which is known to introduce checkerboarding artifacts but is approximately 2× faster to infer than bilinear upsampling followed by learned convolution [3]. To correct for these artifacts, we use joint-bilateral upsampling [1] on the screen-space perspective view as we accumulate the alpha layers and blend the RGB inputs within our real-time renderer. This successfully removes some artifacts.

Bilateral filters are computationally expensive, yet this approach is possible because of our combined hardware and software design: 1) We use a dedicated GPU for inference, which we wish to run as fast as possible for real-time video, and so we make a trade off in the quality of the model inference because 2) We use a dedicated GPU for rendering; rendering the multi-sphere representation is fast, and so we have spare compute capacity on the render card for this filtering. In a setting with a less powerful machine, the filtering could be skipped.

2.3 Network Architecture and Hyperparameters

We include a complete layer description of our network architecture in Table 1. We also include a table of all of our architecture and system hyperparameters (Table 2).

Table 1. U-Net-style convolutional neural network architecture for our approach, as shown in Figure 2 of the main paper. '**k**' is the kernel size, '**s**' is the stride, '**d**' is the dilation factor of the kernel, and '**chns**' is the number of input/output channels (kernels). The network input are the left and right sphere sweep volumes $\mathcal{S}^{\mathrm{L}}$ and $\mathcal{S}^{\mathrm{R}}$. The internal convolutional layers are identical to that of the architecture in [4], except that input to each convolutional layer is also concatenated with the V coordinate ('+1' channel), as described in the main paper. As in [4], each convolutional layer is followed by layer normalization and ReLU non-linearity, except for the last layer. The double-plane sweep baseline uses the same architecture, without additional coordinate channels.

Layer	k	s	d	chns	in	out	input
conv1_1	3	1	1	2×32×3+1/64	1	1	$\mathcal{S}^{\mathrm{L}} + \mathcal{S}^{\mathrm{R}} + V$
conv1_2	3	2	1	64+1/128	1	2	conv1_1 + V
conv2_1	3	1	1	128+1/128	2	2	conv1_2 + V
conv2_2	3	2	1	128+1/256	2	4	conv2_1 + V
conv3_1	3	1	1	256+1/256	4	4	conv2_2 + V
conv3_2	3	1	1	256+1/256	4	4	conv3_1 + V
conv3_3	3	2	1	256+1/512	4	8	conv3_2 + V
conv4_1	3	1	2	512+1/512	8	8	conv3_3 + V
conv4_2	3	1	2	512+1/512	8	8	conv4_1 + V
conv4_3	3	1	2	512+1/512	8	8	conv4_2 + V
conv5_1	4	.5	1	1024+1/256	8	4	conv4_3 + conv3_3 + V
conv5_2	3	1	1	256+1/256	4	4	conv5_1 + V
conv5_3	3	1	1	256+1/256	4	4	conv5_2 + V
conv6_1	4	.5	1	512+1/128	4	2	conv5_3 + conv2_2 + V
conv6_2	3	1	1	128+1/128	2	2	conv6_1 + V
conv7_1	4	.5	1	256+1/64	2	1	conv6_2 + conv1_2 + V
conv7_2	3	1	1	64+1/64	1	1	conv7_1 + V
conv7_3	1	1	1	64+1/67	1	1	conv7_2 + V

Table 2. Hyperparameters for our dataset generation and training.

Parameter	Value
Training / Test target views	63,000 / 27,000
Learning rate	0.0002
Batch size	1
Epochs	4
Optimizer	Adam with $\beta = 0.9$
ODS baseline	0.064 metres
MSI radii	[1, 100] metres
Target view offset (x, y, z)	[0.02, 0.36] metres
Temporal rotation (θ, ϕ, ψ)	$\pm 1.7°$
Temporal translation (x, y, z)	± 0.01 metres
λ_{L2}, $\lambda_{\mathrm{ERP\text{-}L2}}$, or λ_{Per}	1
λ_{TI}	10

2.4 Rendering Pseudocode

In Algorithm 1, we include pseudocode for the Render function from Section 3.3 in the main paper, which is used to train our network architecture. Then, in Algorithm 2, we provide pseudocode for our real-time MSI renderer implemented in Unity, which additionally uses high-resolution video input and a joint-bilateral filter to improve quality.

Algorithm 1: Render function from Section 3.3 (main paper)

Render
  Input:
  $\mathcal{M}$: A $w \times h \times N$ MSI.
  P: A 4×4 matrix representing a target pose.

  Output:
  $\hat{I}$: Rendered ERP image from the target pose.

  foreach pixel location $(u, v) \in I$ **do**
    $\mathbf{r} \leftarrow \text{GetRay}(u, v, P)$
    $\{\mathbf{p}_i\}_{i=1}^{N} \leftarrow \text{GetIntersections}(\mathbf{r}, \mathcal{M})$
    $\{\mathbf{c}_i\}_{i=1}^{N}, \{\alpha_i\}_{i=1}^{N} \leftarrow \text{Sample}(\mathcal{M}, \{\mathbf{p}_i\}_{i=1}^{N})$
    $\hat{I}(u, v) \leftarrow \text{OverComposite}(\{\mathbf{c}_i\}_{i=1}^{N}, \{\alpha_i\}_{i=1}^{N})$
  end
  return $\hat{I}$
end

References

[1] Kopf, J.: 360° video stabilization. ACM Trans. Graph. **35**(6), 195:1–9 (2016). https://doi.org/10.1145/2980179.2982405 6

[2] Serrano, A., Kim, I., Chen, Z., DiVerdi, S., Gutierrez, D., Hertzmann, A., Masia, B.: Motion parallax for 360° RGBD video. TVCG **25**(5), 1817–1827 (2019). https://doi.org/10.1109/TVCG.2019.2898757 2

[3] Sugawara, Y., Shiota, S., Kiya, H.: Super-resolution using convolutional neural networks without any checkerboard artifacts. In: ICIP. pp. 66–70 (2018). https://doi.org/10.1109/ICIP.2018.8451141 6

[4] Zhou, T., Tucker, R., Flynn, J., Fyffe, G., Snavely, N.: Stereo magnification: Learning view synthesis using multiplane images. ACM Trans. Graph. **37**(4), 65:1–12 (2018). https://doi.org/10.1145/3197517.3201323 7

Algorithm 2: Real-time rendering pipeline in Unity

RTRender

Input:
I_L: A high-resolution $w \times h$ left ODS image.
I_R: A high-resolution $w \times h$ right ODS image.
P: Pose of the VR headset

Output:
I: Rendered perspective image from headset pose

$I'_L, I'_R \leftarrow \text{AntialiasedDownsample}(I_L, I_R)$
$\mathcal{M} \leftarrow \text{InferMSI}(I'_L, I'_R)$
$\mathcal{M}' \leftarrow \text{JointBilateralUpsample}(\mathcal{M}, I_L, I_R)$
$\mathcal{S} \leftarrow \emptyset$
foreach layer $l \in \mathcal{M}'$ **do**
  $\mathcal{S}_l \leftarrow \text{TextureSphere}(\mathcal{M}', I_L, I_R, l)$
end
$I \leftarrow \text{RasterizeWithAlpha}(\mathcal{S}, P)$
return I

end