1000FPS+ Novel View Synthesis from End-to-End Opaque Triangle Optimization

Supplementary Material

1. Supplementary Video

We include a supplementary video, which contains the training process visualization of our OTO method, and a few videos showing the downstream applications mentioned in the main paper.

2. Implementation Details

2.1. Initialization

As described in the main paper, one triangle is initialized for one point in the sparse point cloud. Each triangle i is initialized from a position $\mathbf{P}_i$, facing $\mathbf{N}_i$ and scale S_i. The position $\mathbf{P}_i$ is the same as the point position. The facing N is derived from the estimated surface normal of the point cloud at the point position, using Open3D[61]. For the MipNeRF [2] dataset, we use the hyper-parameter of radius 1 and maximum nearest neighbor of 300. The scale $S = \min(S_{i,nn}, S_{i,ceil})$ takes the value of the distance to the nearest neighbor $\mathbf{S}_{i,nn}$, clamped with the maximum scale $S_{i,ceil} = \min_{j\in train\_cam}(||\mathbf{P}_i - \mathbf{P}_j||_2) \times \frac{S_{px}}{f_j}$ using the distance to the closest training camera with focal length f_j and a maximum pixel size S_{px} set as the $1/10$ of image width.

After these parameters are calculated for each triangle, the two control vectors $\mathbf{V}_{i,x}$ and $\mathbf{V}_{i,y}$ of the plane perpendicular to the triangle facing $\mathbf{N}_i$ are determined, by $\mathbf{V}_{i,x} = \mathbf{N}_i \times [0,0,1]^T$ and $\mathbf{V}_{i,y} = \mathbf{N}_i \times \mathbf{V}_{i,x}$. Finally, the exact vertex positions are determined by

$$\mathbf{V}_i = \{[-\frac{1}{2}, 0, \frac{1}{2}]\mathbf{V}_{i,x}\,[-\frac{1}{2\sqrt{3}}, \frac{1}{\sqrt{3}}, -\frac{1}{2\sqrt{3}}]\mathbf{V}_{i,y}\}\cdot S_i + \mathbf{P}_i \tag{6}$$

2.2. Skybox

Due to the sparse point cloud initialization, the early stage of OTO training can be unstable due to the lack of background points. We found that the skybox is essential to prevent the foreground triangles from being overly enlarged, which can be difficult to restore due to the high overlap between these large triangles. Hence, a set of skybox triangles is included in a six-face cube surrounding the scene.

The cube is initialized at the center of the scene, with a scale of $S_{skybox} = \lambda(\max(\mathbf{P}) - \min(\mathbf{P}))$, where λ takes the value of 5 in our experiment to ensure that the entire scene is enclosed in the skybox. For each face of this cube, we first cut it into a square grid of pre-determined resolution, which is of value 50×50 in our experiment. Each square in the grid is cut into two triangles to form the skybox triangles. Please note that the triangles share the same vertex if their vertex is at the same position, to ensure smooth color interpolation. All skybox triangles have optimization diffuse colors, no spherical harmonic parameters for view-dependent colors, and their vertex positions are fixed, which are standard for skybox optimization.

2.3. Rendering

As mentioned in the main paper, the image is divided into tiles of 16×16 size for faster parallel rendering, similar to 3DGS [20]. Unlike 3DGS, all tiled triangle parts are rasterized in parallel instead of from a front-to-back order. No sorting is required either since a depth buffer is used. During backpropagation, this tile is dilated beyond the actual size of the triangle to incorporate the 2D SDF approximation. Our experiment set the temperature $\tau = 10$, so one-pixel dilation is enough.

To calculate the gradient from the two-layer occlusion approximation, a second-layer depth buffer is used to determine the color of the second layer. Two layers of depth buffers are calculated in sequence, to ensure atomized operations are handled without racing.

2.4. Training

Our training uses Adam [21] optimizer for $20,000$ iterations. The base learning rate of vertex positions, colors, specular coefficients, and skybox colors are set to $6\times10^{-4}, 1\times10^{-3}, 1\times10^{-3}, 1\times10^{-2}$ respectively. The vertex position learning rate should be scaled based on image size and scene size when adapted to new datasets.

The density control is employed with a predetermined schedule. Splitting is performed from the start to $15,000$ iterations with an interval of $1,000$ iterations before the total triangle count reaches $60,000$. Pruning small triangles starts at $3,500$ iteration and ends at $20,000$ iterations with an interval of $1,000$ iterations. starts at $3,500$ iteration and ends at $10,000$ iterations with an interval of $1,000$ iterations. Insertion starts at $2,500$ iteration and ends at $10,000$ iterations with an interval of 500 iterations before the triangle counts reach $60,000$.

3. Downstream Application Details

We provide two types of export modes of our trained model to be used in downstream applications, which are model with vertex color and model with textures. The model with vertex color can be imported into OpenGL [51], Unity [45], and UnrealEngine [12] using a standard material that supports vertex color. The drawback of the vertex color model is the lack of support for view-dependent colors learned using spherical harmonics. However, in most downstream applications, the view-dependent colors are usually a result of relighting from environment light, so this is not a significant issue in most cases.

If view-dependent colors need to be exported as well, we provide the export mode with textures, which contain both the diffuse color and the spherical harmonic parameters. However, the settings in the graphics engines need to be handled very carefully to avoid any artifact caused by mipmaps and texture rescaling. This also requires a simple shader code to support spherical harmonics, which is very standard as well.

For better visualizations of the downstream applications, please refer to the supplementary video, where we implement some basic scenarios and effects using Unity [45]. We include variable materials and light sources to demonstrate the capability of relighting. To demonstrate the physical simulation, we added some objects with collider boxes and a character that can walk and jump within the scene freely. We also include a scene with an effect ball that can "burn through" the reconstructed model while adding fire and smoke particle effects for it. These are very standard use cases and special effects and serve as a simple illustration of how our reconstructed model is natively compatible with graphics engine applications.

4. Per-Scene Quantitative Results

We report the per-scene quantitative results on the Mip-NeRF 360 dataset [2], for our full model(Tab. 3), model without 2D SDF approximation(Tab. 4), model without two-layer occlusion approximation(Tab. 5), model without SH view-dependent colors(Tab. 6), model without splitting density control(Tab. 7), model without pruning density control(Tab. 8), and model without insertion density control(Tab. 9).

Table 3. Per-Scene Quantitative Results of OTO on MipNeRF Dataset[2]

scene name	PSNR	SSIM	LPIPS
stump	23.72	0.592	0.387
room	29.44	0.879	0.210
treehill	21.29	0.492	0.455
flowers	19.87	0.469	0.414
counter	26.95	0.824	0.214
bonsai	29.85	0.890	0.187
garden	25.44	0.739	0.238
bicycle	23.13	0.580	0.383
kitchen	28.23	0.854	0.174
average	25.32	0.702	0.296

Table 4. Per-Scene Quantitative Results of OTO without 2D SDF approximation on MipNeRF Dataset[2]

scene name	PSNR	SSIM	LPIPS
stump	19.47	0.272	0.577
room	19.05	0.387	0.622
treehill	17.14	0.211	0.602
flowers	15.54	0.156	0.604
counter	16.75	0.248	0.643
bonsai	16.12	0.230	0.651
garden	18.63	0.243	0.569
bicycle	17.56	0.211	0.600
kitchen	16.42	0.220	0.629
average	17.41	0.242	0.611

Table 5. Per-Scene Quantitative Results of OTO without two-layer occlusion approximation on MipNeRF Dataset[2]

scene name	PSNR	SSIM	LPIPS
stump	19.02	0.321	0.551
room	17.52	0.460	0.579
treehill	14.23	0.208	0.631
flowers	14.13	0.162	0.629
counter	14.95	0.282	0.627
bonsai	15.17	0.277	0.635
garden	17.69	0.305	0.566
bicycle	15.85	0.210	0.625
kitchen	15.67	0.264	0.641
average	16.03	0.277	0.609

Table 6. Per-Scene Quantitative Results of OTO without SH view-dependent colors approximation on MipNeRF Dataset[2]

scene name	PSNR	SSIM	LPIPS
stump	24.10	0.601	0.386
room	29.35	0.876	0.216
treehill	21.42	0.486	0.471
flowers	20.10	0.460	0.427
counter	26.05	0.792	0.252
bonsai	28.59	0.874	0.203
garden	24.63	0.688	0.283
bicycle	22.91	0.558	0.404
kitchen	26.96	0.807	0.216
average	24.90	0.683	0.318

Table 7. Per-Scene Quantitative Results of OTO without splitting density control approximation on MipNeRF Dataset[2]

scene name	PSNR	SSIM	LPIPS
stump	22.56	0.490	0.503
room	28.58	0.859	0.253
treehill	21.63	0.494	0.523
flowers	18.94	0.365	0.541
counter	26.55	0.808	0.245
bonsai	28.64	0.867	0.242
garden	24.49	0.659	0.353
bicycle	22.01	0.489	0.502
kitchen	27.23	0.832	0.210
average	24.51	0.651	0.375

Table 8. Per-Scene Quantitative Results of OTO without pruning density control on MipNeRF Dataset[2]

scene	psnr	ssim	lpips
stump	23.93	0.588	0.395
room	28.73	0.859	0.247
treehill	21.30	0.477	0.487
flowers	19.97	0.445	0.439
counter	26.17	0.788	0.268
bonsai	27.77	0.855	0.239
garden	24.86	0.682	0.305
bicycle	22.81	0.542	0.428
kitchen	26.58	0.794	0.241
average	24.68	0.670	0.339

Table 9. Per-Scene Quantitative Results of OTO without insertion density control on MipNeRF Dataset[2]

scene name	PSNR	SSIM	LPIPS
stump	23.77	0.586	0.405
room	28.39	0.869	0.218
treehill	21.20	0.472	0.496
flowers	19.80	0.448	0.447
counter	26.71	0.820	0.221
bonsai	28.29	0.874	0.209
garden	25.41	0.734	0.242
bicycle	22.97	0.563	0.401
kitchen	28.12	0.850	0.175
average	24.96	0.690	0.313