LightOctree: Lightweight 3D Spatially-Coherent Indoor Lighting Estimation

Supplementary Material

1. Details in Rendering Layer

In this section, we delve into the rendering layer employed within our framework. Leveraging a multi-level octree for sparse illumination representation, we have devised a fast differentiable cone tracing method to render 2D panoramic environment maps from lighting octree.

In classical volume rendering, we calculate the light radiation $L(s)$ from a viewpoint $\mathbf{x}$ in direction ω using the rendering equation (Equation 1). The radiance function is defined as the integral $L(s)$ of transmittance $T(t)$, density $\sigma(t)$, and radiance intensity $L_e(t)$ over the path length s.

$$L(s) = \int_{t=0}^{s} T(t)\sigma(t)L_e(t)\mathrm{d}t \tag{1}$$

To solve this equation, we discretize it with a sampling distance δ_k between consecutive sampling points:

$$L(s) = \sum_{n=1}^{N} C_n T_n \left(1 - e^{-\sigma_n \delta_n}\right), \quad T_n = e^{-\sum_{k=1}^{n-1} \sigma_k \delta_k} \tag{2}$$

We optimize the rendering process with cone tracing to reduce samples and enhance sampling efficiency, inspired by prior works [1, 3]. This approach also effectively integrates multi-scale outputs from the lighting estimation network, adapting to the characteristics of octree-based networks.

Given the angle θ of the cone, the sampling position s_n and sampling step distance δ_n of each sampling point are computed based on the distance traveled and the angle of the previous point (Equation 3). This approach allows for more frequent sampling of nearby light sources, enhancing sensitivity to critical lighting information.

$$s_n = \sum_{k=0}^{n} \delta_k, \quad \delta_n = \begin{cases} c_0, & n = 0 \\ c \cdot s_{n-1} \tan\theta, & n \geq 1 \end{cases} \tag{3}$$

Then, by calculating the cone radius at the sampling point location, we sample octree nodes at depth d_n:

$$d_n = \lceil \log_2 \frac{\delta_n}{l_0} \rceil \tag{4}$$

where l_0 illustrate the minimum side length of leaf node in the octree. Based on these processes, the new rendering equation can be formulated as equation 5.

$$L(s) = \sum_{n=1}^{N} w_n C_n^{d_n} T_n \left(1 - e^{-\sigma_n^{d_n} \delta_n}\right) \tag{5}$$

where $C_n^{d_n}$ and $\sigma_n^{d_n}$ represents the radiance and density of sampling point n at a depth of d_n. And $w_n = \delta_n / s_n$ is the weight for each sampling point.

To ensure the differentiability of the above process, we derive the derivatives of the color and opacity with respect to the forward rendering process, as shown in equation 6, where $E_n = 1 - e^{-\sigma_n^{d_n} \delta_n}$ to simplify the formula.

$$\frac{\partial L(\mathbf{x},\omega)}{\partial \sigma_t^{d_t}} = w_t C_t^{d_t} T_{t+1} \delta_t - \left[L(\mathbf{x},\omega) - \sum_{n=1}^{t} w_n C_n^{d_n} T_n E_n \right] \delta_t \tag{6}$$

$$\frac{\partial L(\mathbf{x},\omega)}{\partial C_t^{d_t}} = \frac{\partial \left(\sum_{n=1}^{N} C_n^{d_n} T_n \left(1 - e^{-\sigma_n^{d_n} \delta_n}\right)\right)}{\partial C_t^{d_t}} = T_t E_t \tag{7}$$

The above equations describe how to calculate derivatives during rendering. The derivatives of color can be computed during the sampling process, while the derivatives of density require the use of the final rendering results. Building upon the derived formulas, we have implemented a differentiable renderer using Taichi[2], enabling rapid forward rendering and gradient calculations.

2. Details in Object Insertion

Given that common object representations are typically meshes, while point clouds are prevalent in AR applications, and our lighting representation employs a voxel octree structure, we've devised a fusion rendering method that seamlessly combines point clouds, meshes, and octrees.

2.1. Scene Representation

Background Representation Differential rendering encompasses the rendering of virtual objects and the shading computation, which requires multiple environment lighting queries. Each query involves casting a cone from the surface position x into the environment to find radiance incident from the ray direction (solid angle direction w_i). We employ a point cloud to represent the original scene, including depth and color information from the viewpoint to the surfaces. This point cloud is per-pixel, allowing us to skip ray collisions from the viewpoint to the scene surfaces and directly use the projected depth from the point cloud as the position. We utilize this information to determine the spatial relationship between the scene and virtual objects. We perform cone tracing to illuminate the rendered virtual objects.

Virtual Object Representation To efficiently perform ray intersections and integrate it into the Taichi framework, we normalizes the bounding boxes of virtual objects to the range of 0 to 1. For each triangular face, a multi-level regular grid space division is performed in this three-dimensional space, which is managed using the SNode structure.

2.2. Differential Rendering

Due to the fact that virtual objects in the scene typically occupy only a small portion, yet they may consist of thousands or even tens of thousands of triangles, achieving real-time rendering with three different data structures in AR applications requires a new approach to index lookup. During the ray traversal through the bounding box, we can calculate the index of the first intersected object grid based on the ray's origin R_o and direction R_d, as shown below:

$$H_{min} = \max\left(\min\left(\frac{Box_{min} - R_o}{R_d}, \frac{Box_{max} - R_o}{R_d}\right)\right)$$
$$I_0 = floor\left(\frac{R_o + R_d H_{min} - Box_{min}}{Box_{size}}\right) \tag{8}$$

where Box_{min} and Box_{max} represent the coordinates of the bounding box's vertices, and Box_{size} represents the size of the bounding box.

In subsequent tracing steps, the index of the next grid can be obtained based on the ray's current position R_t and normalized forward direction R'_d as follows:

$$\mathrm{R}'_t = (1 - R_t) * (R'_d \geq 0) + R_t * (R'_d < 0)$$
$$\mathrm{I}_{t+1} = I_t + \mathrm{G}\left(\frac{R'_t}{R'_d}\right) * \mathrm{sign}\left(R'_d\right) \tag{9}$$

where $G(\cdot)$ calculates the axis with the smallest advancement in the axis directions. For example, if $G((0.5, 0.2, 0.7)) = (0, 1, 0)$. During rendering, we emit a ray from the camera viewpoint for each pixel on the screen and employ ray tracing to trace it until it intersects with an object. Upon detecting an intersection, we emit a cone of rays from the object's surface in various directions based on the surface normal. This cone of rays is then traced and rendered using ray marching, with the results obtained from volume rendering serving as the illumination for surface shading. Same to the rendering layer, we begin with a minimum step size δ_{min} and later perform long-distance mip-map sampling based on the cone's radius.

As the scene is managed using an octree, obtaining the voxel index directly from the sampling point's position is not feasible. Therefore, we trace the voxel recursively based on the current sampling point's location within the volume. To improve performance, we utilize a shared recursive stack for all cone rays, modifying the sampling order based on the ray's direction. Instead of clearing the stack after each sampling, we continue the recursion in the next sampling, enhancing efficiency. Once we determine the sampling point's position and its associated node, we perform trilinear interpolation using the values within the node block, accumulate the results along the ray and apply for shading.

3. Details in Dataset Construction

We train our model using photorealistic renderings of indoor scenes from the FutureHouse synthetic dataset[4]. This dataset consists of artist-designed indoor panoramas with high-quality geometry and HDR environment maps. We can extract photographs from this dataset to obtain input/output pairs for training. It is worth mentioning that we chose to use panorama datasets instead of InteriorNet dataset used by [5] and [12], or the OpenRoom dataset used by [9] and [10]. This is because the images in InteriorNet have low dynamic range, and the resolution of the panoramic data in OpenRoom is too low. These shortcomings make these two datasets less ideal for lighting estimation tasks. Although InteriorVerse proposed by [11] is a better choice, at the time of writing this paper, the lighting data of this dataset are still unavailable. Fortunately, with the HDR panoramas and related geometry information provided by FutureHouse, we can construct the training data that meets our needs.

Figure 1. Schematic visualization of the constructed dataset.

Each training sample $\{I, D, O_s, \{I_{nv}, P_{nv}\}_N\}$ contains one LDR perspective image I as inputs and paired data as ground truth to supervise depth and lighting voxel octree. The construction process of a single sample is as follows: For a perfect 360-degree camera, we can treat its imaging plane as a sphere, and any point in world coordinates can be projected onto this sphere as a spherical projection. Therefore, for each RGB panoramic image and its corresponding depth panoramic image in the dataset, we can reproject all points onto the world coordinate system using the inverse transformation of spherical projection, thus obtaining the 3D point cloud of the scene $\mathcal{P}$. Based on the 3D point cloud $\mathcal{P}$, we can construct all the required ground truth. For the input image I and its corresponding depth map D, we use a perspective camera model with a field of view (FOV) of 90 degrees and an aspect ratio of 1 (similar to what [6] did) to reproject points on the sphere onto a cube face, obtaining four perspective images that cover a 360-degree horizontal view. For the ground truth of the scene's voxel octree, we can directly construct it from the point cloud $\mathcal{P}$ using the method introduced by [7, 8]. For the new viewpoint images

$\{I_{nv}, P_{nv}\}_N$, we construct a uniform voxel grid with a resolution of 256^3 from the point cloud $\mathcal{P}$, and then render N HDR panoramic images at locations that are visible in the perspective input image I using a rendering method similar to NeRFs. The corresponding camera positions P_{nv} are also recorded during rendering. Based on the 28,579 panoramic views from 1,752 house-scale scenes provided by Future-House, we construct and select 113,232 pairs of data for model training. We use 90% (1,570) of the scenes to train our model and reserve 10% (180) for evaluation.

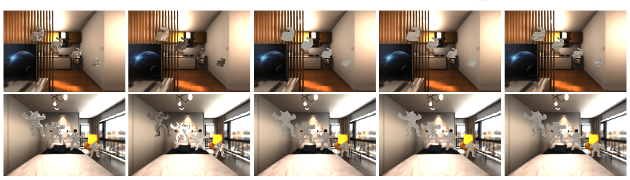
4. Additional Details

Bad cases. Depth estimation inaccuracies may lead to unforeseen outcomes in environments with complex lighting or geometry, as depicted in Figure 2. Artifacts observed in both Wang et al. and ours are presumably due to erroneous initial depth compromising alpha prediction accuracy. However, the distant environmental illumination assumption may also fail in these scenarios with intense spatially-varying lighting.

(a) Gardner et al. [21] (b) Li et al. [37] (c) Wang et al. [42] (d) Ours (e) Reference Image

Figure 2. Cases of failure in certain special circumstances.

Data used in user study. Images are showcased in Figure 3, where video frames in virtual object insertion results were extracted and amalgamated into a single composite image.



(a) Gardner et al. [21] (b) Li et al. [37] (c) Lighthouse [24] (d) Ours (e) Input Image

Figure 3. Virtual object insertion examples used in user study.

References

[1] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, Elmar Eisemann, and MLeal Llaguno. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum*, page 1921–1930, 2011. 1

[2] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019. 1

[3] Samuli Laine and Tero Karras. Efficient sparse voxel octrees. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 55–63, 2010. 1

[4] Zhen Li, Lingli Wang, Xiang Huang, Cihui Pan, and Jiaqi Yang. Phyir: Physics-based inverse rendering for panoramic indoor images. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 12703–12713. IEEE, 2022. 2

[5] Pratul P. Srinivasan, Ben Mildenhall, Matthew Tancik, Jonathan T. Barron, Richard Tucker, and Noah Snavely. Lighthouse: Predicting lighting volumes for spatially-coherent illumination. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 8077–8086. Computer Vision Foundation / IEEE, 2020. 2

[6] Fu-En Wang, Hou-Ning Hu, Hsien-Tzu Cheng, Juan-Ting Lin, Shang-Ta Yang, Meng-Li Shih, Hung-Kuo Chu, and Min Sun. Self-supervised learning of depth and camera motion from 360 videos. In *Asian Conference on Computer Vision*, pages 53–68. Springer, 2018. 2

[7] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Transactions on Graphics*, page 1–11, 2018. 2

[8] Peng-Shuai Wang, Yang Liu, and Xin Tong. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics*, page 1–15, 2022. 2

[9] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 2472–2481. Computer Vision Foundation / IEEE, 2020. 2

[10] Zhengqin Li, Ting-Wei Yu, Shen Sang, Sarah Wang, Meng Song, Yuhan Liu, Yu-Ying Yeh, Rui Zhu, Nitesh B. Gundavarapu, Jia Shi, Sai Bi, Hong-Xing Yu, Zexiang Xu, Kalyan Sunkavalli, Milos Hasan, Ravi Ramamoorthi, and Manmohan Chandraker. Openrooms: An open framework for photorealistic indoor scene datasets. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 7190–7199. Computer Vision Foundation / IEEE, 2021. 2

[11] Jingsen Zhu, Fujun Luan, Yuchi Huo, Zihao Lin, Zhihua Zhong, Dianbing Xi, Jiaxiang Zheng, Rui Tang, Hujun Bao, and Rui Wang. Learning-based inverse rendering of complex indoor scenes with differentiable monte carlo raytracing. In *SIGGRAPH Asia*, 2022. 2

[12] Zian Wang, Jonah Philion, Sanja Fidler, and Jan Kautz. Learning indoor inverse rendering with 3d spatially-varying lighting. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12518–12527. IEEE, 2021. 2