Vertex Features for Neural Global Illumination

RUI SU, School of Computer Science, Peking University, China
HONGHAO DONG, School of Computer Science, Peking University, China
HAOJIE JIN, School of Computer Science, Peking University, China
YISONG CHEN, School of Computer Science, Peking University, China
GUOPING WANG*, School of Computer Science, Peking University, China
SHENG LI*, School of Computer Science, Peking University, China

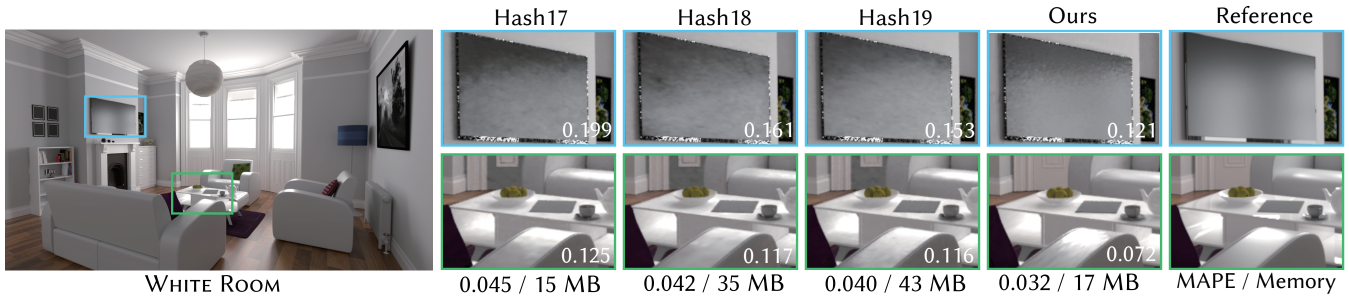

Fig. 1. *Neural vertex features* (Ours) leverages the geometric priors in neural rendering applications, effectively culling the redundant features by focusing on the occupancy of the scenes. Compared to the multi-resolution hashing encoding [Müller et al. 2022] at different sizes of hash tables (2^{17}-2^{19} features each level), our method achieves better scalability and significantly less memory overhead at the same level of visual quality. The better compactness of our method also results in faster inference for being more friendly to the cache hierarchies of modern GPUs.

Recent research on learnable neural representations has been widely adopted in the field of 3D scene reconstruction and neural rendering applications. However, traditional feature grid representations often suffer from a substantial memory footprint, posing a significant bottleneck for modern parallel computing hardware. In this paper, we present neural vertex features, a generalized formulation of learnable representation for neural rendering tasks involving explicit mesh surfaces. Instead of uniformly distributing neural features throughout 3D space, our method stores learnable features directly at mesh vertices, leveraging the underlying geometry as a compact and structured representation for neural processing. This not only optimizes memory efficiency, but also improves feature representation by aligning compactly with the surface using task-specific geometric priors. Additionally, neural vertex features offer improved feature representation by compactly aligning with the surface using task-specific geometric priors. We validate our neural representation across diverse neural rendering tasks, with a specific emphasis on neural radiosity. Experimental results demonstrate that our method reduces memory consumption to only one-fifth (or even less) of grid-based representations, while maintaining comparable rendering quality and lowering inference overhead.

*corresponding author.
Project website: https://woaixuexisr.github.io/papers/neural-vertex-features.

CCS Concepts: • **Computing methodologies → Ray tracing**; **Neural networks**.

Additional Key Words and Phrases: Neural Rendering, Encodings, Global Illumination, Radiosity

ACM Reference Format:
Rui Su, Honghao Dong, Haojie Jin, Yisong Chen, Guoping Wang, and Sheng Li. 2025. Vertex Features for Neural Global Illumination. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3757377.3763826

1 INTRODUCTION

Global illumination has become an important topic of research in computer graphics, while recent research has leveraged the power of deep learning to achieve realistic global illumination effects. Various methods have been developed to tackle the challenge, such as neural rendering [Coomans et al. 2024; Hadadan et al. 2021; Müller et al. 2021; Su et al. 2024]. These techniques generally follow a pipeline where spatial features are extracted through input encoding and then processed by multi-layer perceptrons (MLPs) to generate the desired outputs. While these approaches have shown remarkable results in producing high-quality images, they often face significant challenges with respect to memory footprint and thus inference overhead. On the other hand, as the parallel computing hardware with tensor arithmetic accelerators iterates, more and more neural workloads tend to become memory-limited where the memory access of neural networks becomes one of the main performance bottlenecks. For example, even the cutting-edge GPUs have an L2 memory of less than 72MB, making commonly used neural networks difficult to fit into caches.

One of the key aspects influencing the performance of neural rendering models is the input encoding. A widely adopted technique is grid-based encoding, such as the multi-resolution hash grid [Müller et al. 2022], which scatters spatial features over structured 3D grids to enable the use of compact MLPs and accelerate convergence. While this approach is memory-efficient compared to fully dense grids due to its hash-based storage, it still allocates features in empty space rather than strictly along the scene surfaces. Even with hash-based storage, in practice, the features used in the grid account for only 55% to 72% of the total hash table capacity, leading to memory inefficiency. This discrepancy may cause unnecessary memory consumption, particularly in real-time applications.

Our method encodes spatial information directly at the vertices of the scene's mesh, leveraging the known geometry as a priori in neural rendering applications, and thus allowing for a more compact and memory-efficient representation of the scene. Unlike multi-resolution grids, which allocate resources to unnecessary space, our approach concentrates computation on the actual scene geometry, significantly reducing memory overhead. Furthermore, we introduce a multi-resolution representation of surface feature encoding that can dynamically refine the mesh-based feature on training loss statistics. This enables our method to capture high-frequency details even in regions that initially have low mesh density.

We evaluate our approach with diverse neural rendering applications like Neural Radiosity [Hadadan et al. 2021] and dynamic neural radiosity (DNR) [Su et al. 2024] using both static and dynamic scenes, as well as neural path guiding [Dong et al. 2023]. Our method demonstrates a significant improvement in memory efficiency, achieving over 5× reduction in memory overhead while maintaining comparable image quality to state-of-the-art methods. By focusing on encoding spatial information directly at the mesh vertices and utilizing adaptive surface subdividing, we address the challenges of high memory consumption and inefficient feature representation in prior approaches.

Overall, our main contributions are as follows:

- We present a novel encoding method, neural vertex features, that stores spatial information at the vertices of the scene's mesh, offering a memory-efficient and computationally effective solution for neural rendering.
- We propose a multi-resolution surface learnable feature that dynamically refines the mesh based on training loss, allowing high-frequency illumination details to be accurately captured even in coarse regions with initially low resolution.
- Several neural rendering approaches validate the effectiveness of our formulation in significantly reducing memory usage while still achieving high-quality rendering results. This advancement may benefit a wide range of neural rendering tasks that previously depended on hash grids, offering enhanced applicability and flexibility.

2 RELATED WORK

Neural Rendering. Deep learning techniques have been widely applied to facilitate rendering applications. Many works focus on image-space techniques, employing deep generative networks for tasks such as denoising [Işık et al. 2021; Vogels et al. 2018], up-scaling [Wu et al. 2023; Zhong et al. 2023], and directly operating in the screen space [Diolatzis et al. 2022; Granskog et al. 2020]. Other works explore object-space techniques, such as [Zheng et al. 2024, 2023], where networks are used to model the light transfer function between individual objects and the environment, subsequently composing them to produce globally illuminated images.

The success of neural scene representations, particularly Neural Radiance Fields (NeRF) [Mildenhall et al. 2021], has further advanced the application of deep learning in photorealistic rendering. These works operate in scene-space, where neural representations model the spatial features of the scene, decoded using lightweight MLPs. Some works [Coomans et al. 2024; Hadadan et al. 2021; Müller et al. 2021; Su et al. 2024] have applied these methods for efficient caching and querying of spatial radiance, while another application focuses on enhancing path tracing by encoding target distributions for path guiding [Dong et al. 2023; Huang et al. 2024]. In this work, we introduce a novel method for encoding spatial features more efficiently, which can further improve these approaches.

Network Encoding. Research has also found that input encoding plays a crucial role in network performance. Early methods, such as [Mildenhall et al. 2021], use frequency encoding to project the 3D coordinate inputs into a high-dimensional space using Fourier basis. Another set of approaches employs trainable encoding with multi-resolution spatial grids [Hadadan et al. 2021] or hash tables [Müller et al. 2022]. The trainable encoding technique reduces the burden on pure MLP methods, resulting in lower computational costs by keeping the MLP size sufficiently small [Takikawa et al. 2023]. However, the feature grid approach is inefficient as it allocates many features to empty space, while the scene surface scales in 2D.

To address this inefficiency, feature-decomposition-based works such as [Cao and Johnson 2023; Fridovich-Keil et al. 2023; Shao et al. 2023] project the dense grid along one or more axes and combine the resulting lower-dimensional features, namely *k-planes*. While these techniques ensure fewer parameters, they make the strong assumption that sparsity in the data can be well explained by axis-aligned projections, leaving the MLP to model the correlation of high dimensionalities. Another approach is learning the indexing function [Li et al. 2023; Takikawa et al. 2022], where an index codebook holds the lookup indices into the feature codebook. Takikawa et al. [2023] combine learned indexing with spatial hashing, arithmetically combining their indices. Additionally, some methods directly generate meshes from neural implicit fields and store features on them, yielding a compact representation that supports both reconstruction and deformation [Bao and Yang et al. 2022; Mahajan et al. 2024].

If the surface of interest is known a priori, compact structures such as octrees [Takikawa et al. 2021] or sparse grids [Chabra et al. 2020; Peng et al. 2020] can be used to cull away unused features. However, these methods primarily work on neural SDF representations for a single object and are not scalable for rendering complex indoor scenes. In this work, we propose a neural vertex feature representation that captures scene spatial features better and is better suited for rendering tasks.

3 BACKGROUND

3.1 Rendering Equation

The fundamental formulation for light transport simulation is the rendering equation [Kajiya 1986], which expresses the outgoing radiance at a point $\mathbf{x}$ in a given direction ω_o as the sum of emitted radiance and the integral of incoming radiance, weighted by the bidirectional reflectance distribution function (BRDF):

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\mathcal{H}^2} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) \left|(\mathbf{n} \cdot \omega_i)\right| \, \mathrm{d}\omega_i. \quad (1)$$

Here, $L(\mathbf{x}, \omega_o)$ denotes the outgoing radiance at point $\mathbf{x}$ in direction ω_o, $L_e(\mathbf{x}, \omega_o)$ is the emitted radiance, and the integral accumulates the incoming radiance $L_i(\mathbf{x}, \omega_i)$ over the hemisphere $\mathcal{H}^2$. The term $f_r(\mathbf{x}, \omega_i, \omega_o)$ is the BRDF that describes the reflectance properties of the surface. The equation is recursive in nature, as the incoming radiance at each point is the outgoing radiance from other points in the scene, making it computationally expensive to solve.

3.2 Neural Radiosity

To address the complexity of solving the rendering equation, Hadadan et al. [2021] introduced neural radiosity, using a neural network with parameters θ to predict outgoing radiance $L(\mathbf{x}, \omega_o)$. The network is optimized by minimizing the residual between the left-hand side (LHS) and the right-hand side (RHS) of the rendering equation:

$$\begin{aligned} r_\theta(\mathbf{x}, \omega_o) &= L_\theta(\mathbf{x}, \omega_o) - L_e(\mathbf{x}, \omega_o) \\ &\quad - \int_{\mathcal{H}^2} f_r(\mathbf{x}, \omega_i, \omega_o) L_\theta(\mathbf{x}'(\mathbf{x}, \omega_i), -\omega_i) \left|(\mathbf{n} \cdot \omega_i)\right| \, \mathrm{d}\omega_i, \end{aligned} \quad (2)$$

where $\mathbf{x}'(\mathbf{x}, \omega_i)$ denotes the closest surface intersection of the ray with origin $\mathbf{x}$ and direction ω_i. The reflected radiance is computed by tracing rays in the reverse direction (RHS rays), and Monte Carlo integration is used to estimate the integral of reflected radiance. The radiance value is predicted by the neural network $L_\theta(\mathbf{x}, \omega_o)$.

This optimization is achieved by minimizing the mean squared error (MSE) loss across multiple points and directions in the scene, as described by the following objective:

$$\theta^* = \arg\min_\theta \int_{\mathcal{S}} \int_{\mathcal{H}^2} r_\theta(\mathbf{x}, \omega_o)^2 \, \mathrm{d}\omega_o \, \mathrm{d}\mathbf{x}, \quad (3)$$

where $\mathcal{S}$ represents the scene surface and $\mathcal{H}^2$ is the hemisphere over each shading point $\mathbf{x}$.

Neural Radiosity effectively leverages the radiometric prior to optimize network parameters and model the global radiance field with lightweight neural networks and trainable spatial feature grids. However, they rely on using multi-resolution feature grids for spatial encoding, which suffers from inefficiencies in memory usage, as large portions of the grid are allocated to empty space. To address these limitations, we propose a more memory-efficient approach by directly encoding neural features at the vertices of the scene's mesh. This eliminates the need for a multi-resolution grid and significantly reduces memory overhead.

4 METHOD

In this section, we introduce neural vertex features and a multi-resolution surface representation to model spatially varying features on the scene manifold $\mathcal{S}$. We then describe our optimization strategy and demonstrate the effectiveness of our method across various neural rendering applications.

4.1 Neural Vertex Features

In neural rendering tasks, network queries are typically performed on the surface of the scene, where the model needs to retrieve and process spatial information to generate realistic lighting, shadows, and textures. Since these queries are inherently tied to the surface, it becomes natural to maintain neural features directly on the scene's surface. The explicit mesh representation of the scene, which already defines the geometry of the surface in terms of vertices and edges, naturally lends itself to this approach.

To this end, we propose neural vertex features, where neural features are stored along with each vertex of the scene's geometries. For simplicity in our experiments, we assume that the geometry consists solely of triangular faces while discussing the potential applications of alternatives, e.g., quadrilaterals. Formally, the union of all scene vertices is represented as $\mathcal{V}$ (including the virtual vertices, which will be discussed later). We maintain a feature vector array $\mathcal{G} \in \mathbb{R}^{n \times d}$, where $n = \left|\mathcal{V}\right|$ is the number of vertices and d is the length of the feature vector. Each vertex $\mathbf{v}_i \in \mathcal{V}$ is uniquely mapped to a corresponding feature vector $\mathbf{g}_i \in \mathcal{G}$ based on its vertex index i.

As illustrated in Fig. 2, for each spatial query $\mathbf{x}$, we first identify the vertices $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ that corresponds to the surface primitive $\mathcal{P}$. Then, we perform barycentric interpolation on the neural features associated with the vertices of the triangle primitive to obtain the feature at the queried point.

$$\mathbf{f}_\mathbf{x} = (1 - u - v) \cdot \mathbf{g}_i + u \cdot \mathbf{g}_j + v \cdot \mathbf{g}_k, \quad (4)$$

where $\mathbf{f}_\mathbf{x}$ is the encoded feature at the spatial point $\mathbf{x}$, and u, v are the barycentric coordinates. The neural features $\{\mathbf{g}_i, \mathbf{g}_j, \mathbf{g}_k\}$ correspond to the vertices $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ of the triangle.

Identification of the corresponding primitive ID and calculation of the barycentric coordinates is straightforward in current rendering frameworks, whether using ray tracing or rasterization, as these operations are commonly performed during the intersection tests or fragment shading stages, with hardly any performance overhead.

Discussion. Neural vertex features can encode spatial information directly at the vertices of the mesh. Compared to the multi-resolution feature grid, storing neural features at the vertices enables a more memory-efficient approach. This strategy exploits the geometric priors to allow for a compact and precise representation of the scene, efficiently capturing important surface details while minimizing unnecessary storage overhead.

4.2 Multi-resolution Surface Feature Encoding

While neural vertex features provide a compact and surface-aligned representation for neural rendering, their effectiveness can be limited by the density of the underlying mesh. In regions where the mesh is coarse or lacks sufficient geometric resolution, the sparse distribution of vertices may hinder the representation of high-frequency illumination details, leading to artifacts or degraded rendering quality, as shown in Fig. 5.

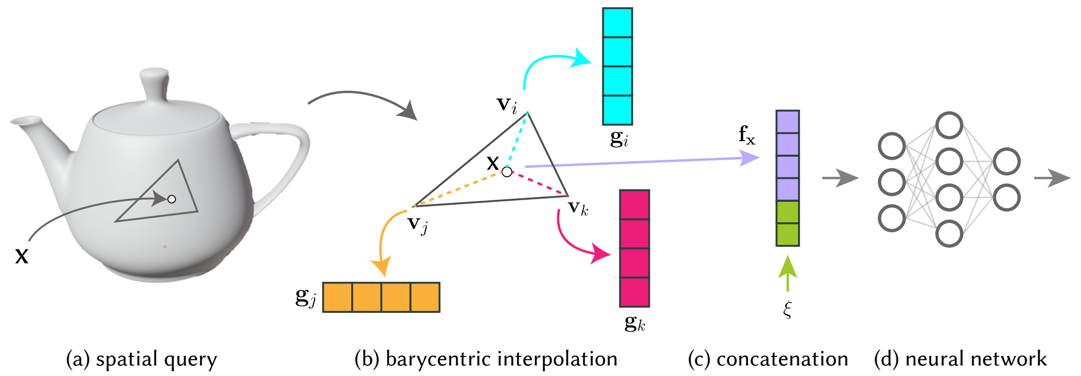

Fig. 2. Schematic of our method. The trainable features are directly attached to the geometry vertices within the scene. To support efficient modeling and querying for arbitrary positions within the domain of scene surfaces $\mathcal{S}$, we use the barycentric coordinate (u, v) of the position $\mathbf{x}$ over the triangle primitive $\mathcal{P}$ to interpolate with the features in each vertex of $\mathcal{P}$, where all these features could be easily acquired in any programmable ray-tracing or rasterization pipelines. The obtained weighted feature $\mathbf{f}_\mathbf{x}$ is then used as network inputs and could be concatenated with other inputs ξ to work with different downstream applications, e.g., neural radiosity [Hadadan et al. 2021]. Implementation details are provided in Sec. 5.

To address this issue, we propose a multi-resolution strategy that increases the density of learnable features in necessary regions without modifying the actual mesh topology. Specifically, we introduce a per-face level of detail (LOD) factor k_i, where $i \in [1, |\mathcal{F}|]$ indexes original triangle faces in the mesh, and $\mathcal{F}$ denotes the set of all faces. The factor k_i specifies that each edge of triangle i is divided into k_i segments, forming a regular triangular grid within the face, as illustrated in Fig. 3. For each resulting virtual vertex, we assign a trainable neural feature, analogous to those defined on the actual mesh vertices.

For each spatial query point $\mathbf{x}$ within an original triangle face i, with barycentric coordinates (u, v), we determine the virtual sub-triangle in which $\mathbf{x}$ resides, and compute the corresponding local barycentric coordinates (u', v') within that virtual triangle as:

$$\begin{aligned} \bar{u} &= \lfloor ku \rfloor, \quad \bar{v} = \lfloor kv \rfloor, \\ \tilde{u} &= ku - \bar{u}, \quad \tilde{v} = kv - \bar{v}, \end{aligned} \tag{5}$$

$$(u', v') = \begin{cases} (\tilde{u}, \tilde{v}) & \text{if } \tilde{u} + \tilde{v} \leq 1, \\ (1 - \tilde{u}, 1 - \tilde{v}) & \text{otherwise.} \end{cases} \tag{6}$$

The virtual sub-triangle that contains $\mathbf{x}$ is identified by the integer grid coordinates $(\bar{u}, \bar{v})$. After this step, we perform the aforementioned barycentric interpolation using the features of the three corresponding virtual vertices.

All LOD factors are initially set to $k_i = 1$, meaning no virtual vertices are added. During training, these factors are adaptively increased in response to the training loss, allocating higher spatial resolution to more challenging regions. When a face is refined, new virtual vertices are generated, and their features are initialized via interpolation from the existing representation. These features replace the originals, allowing the network to refine its capacity where needed. The loss-based update strategy is described in the next section.

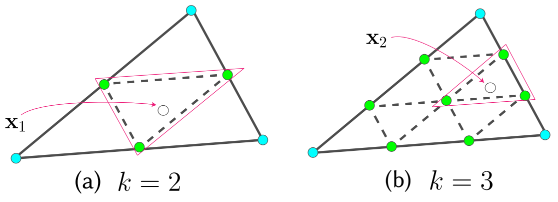

Fig. 3. Illustration of our multi-resolution feature encoding strategy. Blue vertices denote original mesh vertices, green vertices are virtual ones, and dashed lines indicate the associated LOD of features. (a) shows $k = 2$, and (b) shows $k = 3$. Query points $\mathbf{x}_1$ and $\mathbf{x}_2$ lie within the subdivided feature representation highlighted by red boxes, enabling denser and more adaptive illumination feature without actually modifying the original mesh.

4.3 Optimization

To optimize the model, we assume that the loss is defined on the scene surface. The per-point loss at $\mathbf{x}$ is denoted as $l_\theta(\mathbf{x})$, where θ represents the parameters of the network. The total loss function of the network is defined as

$$\mathcal{L}(\theta) = \int_{\mathcal{S}} l_\theta(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{7}$$

where $\mathcal{S}$ represents the union of all surfaces in the scene.

To estimate the total loss, we use Monte Carlo integration. Previous work [Hadadan et al. 2021; Su et al. 2024] typically employs a probability distribution proportional to the surface area to sample positions on the scene surface. However, in our model, the features are not uniformly distributed across the scene surface, leading to inefficient training. Specifically, areas with lower feature density tend to converge faster than regions with higher feature density, potentially slowing down overall optimization. To address this issue, we adopt an importance sampling strategy that initializes the sampling density based on the current feature distribution and dynamically adjusts it in sync with the evolving LOD levels.

We define the sampling probability at the triangle level, incorporating both area-based and vertex-based strategies. Specifically, let $p(i)$ denote the sampling probability of triangle i, where $i \in [1, |\mathcal{F}|]$

indexes the original mesh faces. The area-based probability $p_{\text{area}}(i)$ and the vertex-based probability $p_{\text{vertex}}(i)$ are computed as:

$$p_{\text{area}}(i) = \frac{A(i)}{A_{\text{total}}}, \quad p_{\text{vertex}}(i) = \frac{N(i)}{N_{\text{total}}}, \tag{8}$$

where $A(i)$ is the area of triangle i, and $N(i)$ is the number of vertices (including both real and virtual vertices) associated with the subdivided mesh M_i, computed as $N(i) = \frac{(k_i+1)(k_i+2)}{2}$. A_{total} and N_{total} denote the total surface area and the total number of vertices across the entire scene, respectively.

The sampling probability $p(i)$ for triangle i is defined as a weighted combination of the area-based and vertex-based probabilities, controlled by a factor α:

$$p(i) = \alpha \cdot p_{\text{area}}(i) + (1-\alpha) \cdot p_{\text{vertex}}(i). \tag{9}$$

During training, we adaptively update the LOD factor k_i to refine regions that require higher resolution. Following prior works [Diolatzis et al. 2022; Zhao and Zhang 2015], we use the training loss statistics to guide this update. Specifically, we compute the average loss $L(i)$ for each triangle i over a training interval and adjust the LOD factor accordingly.

Let L'_{mean} and L'_{std} denote the mean and standard deviation of the adjusted loss values $L(i)'$, where

$$L(i)' = L(i) \cdot \log(R(i) + 1), \tag{10}$$

with $L(i)$ and $R(i)$ representing the loss and radiance of triangle i, respectively. The LOD factor is then updated as

$$k_i' = k_i + \left\lfloor \frac{L(i)' - L'_{\text{mean}}}{L'_{\text{std}}} \right\rfloor, \quad \text{if } L(i)' > L'_{\text{mean}} + 2 \cdot L'_{\text{std}}, \tag{11}$$

and remains unchanged otherwise.

4.4 Rendering Task-specific Formulation

For different tasks, it is necessary to select an appropriate per-point loss (Eq. 7). In this work, we apply our method to a neural rendering task, where the network directly outputs the radiance $L_\theta(\mathbf{x}, \omega_o)$ at a given point $\mathbf{x}$ in the scene and for a given outgoing direction ω_o. Specifically, we follow the Neural Radiosity method [Hadadan et al. 2021], where the residual of the rendering equation (Eq. 2) is used as the training target to optimize the model.

Static Scene. On static scenes, the per-point loss is the residual of the render equation at point $\mathbf{x}$, which integrates on the hemisphere for ω_o.

$$l_\theta(\mathbf{x}) = \int_{\mathcal{H}^2} r_\theta(\mathbf{x}, \omega_o)\, \mathrm{d}\omega_o, \tag{12}$$

where the $r_\theta(\mathbf{x}, \omega_o)$ is the residual of the rendering equation and $\mathcal{H}^2$ is the hemisphere space.

Evaluation of the per-point loss is performed using Monte Carlo integration. Specifically, we uniformly sample the outgoing direction ω_o^j within the upper hemisphere of point $\mathbf{x}$. The network is then queried to obtain the radiance $L_\theta(\mathbf{x}, \omega_o^j)$. For the right-hand side of the rendering equation, we sample the reflection direction $\omega_i^{j,k}$ and intersect it with the scene at $\mathbf{x}'(\mathbf{x}, \omega_i^{j,k})$. The network is queried again to obtain the radiance $L_\theta(\mathbf{x}'(\mathbf{x}, \omega_i^{j,k}), -\omega_i^{j,k})$. This process is repeated for M samples, and the average value is taken as the result:

$$\begin{aligned} r_{\theta,MC}(\mathbf{x}, \omega_o^j) &= L_\theta(\mathbf{x}, \omega_o^j) - L_e(\mathbf{x}, \omega_o^j) \\ &- \frac{1}{M}\sum_{k=1}^{M} \frac{f_r(\mathbf{x}, \omega_o^j, \omega_i^{j,k}) L_\theta(\mathbf{x}'(\mathbf{x}, \omega_i^{j,k}), -\omega_i^{j,k}) \left|\mathbf{n} \cdot \omega_i^{j,k}\right|}{p(\omega_i^{j,k})}, \end{aligned} \tag{13}$$

where $p(\omega_i^{j,k})$ is the sampling weight according to the BSDF importance sampling.

Dynamic Scene. For dynamic scenes, we follow the scene definition as described in [Su et al. 2024]. The scene consists of n animated components, each of which is predefined with starting and ending states. A scalar value $v_i \in [0, 1]$ is used to interpolate the states for each component. The scene state can be described by these values, referred to as the *explicit scene vector* $\mathbf{v} \in \mathbb{R}^n$. The per-point loss is also an integral over the entire animation space $\mathcal{V}$.

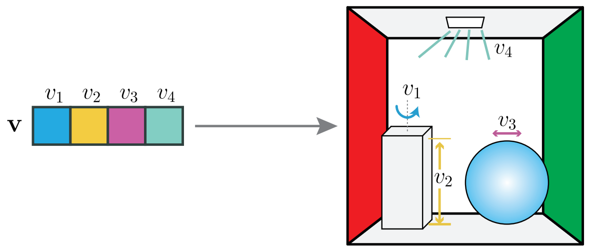

Fig. 4. Illustration of dynamic scene representation. To flexibly represent a dynamic scene, we use a scalar to encode the *state* of each animated component/node. The encoded domains could include the deformation (v_2), rigid body transformation (v_1, v_3), and lighting condition (v_4), for example. Together, these scalar values v_i constitute the *explicit scene state* $\mathbf{v}$.

$$l_\theta(\mathbf{x}) = \int_{\mathcal{V}} \int_{\mathcal{H}^2} r_\theta(\mathbf{x}, \omega_o, \mathbf{v})\, \mathrm{d}\omega_o\, \mathrm{d}\mathbf{v}, \tag{14}$$

where $r_\theta(\mathbf{x}, \omega_o, \mathbf{v})$ is the residual of the rendering equation conditioned on the scene state $\mathbf{v}$. The evaluation of the per-point loss is similar to the static scene, with the additional step of sampling over the scene state $\mathbf{v}$.

The model should also change slightly, and we should model the radiance $L_\theta(\mathbf{x}, \omega_o, \mathbf{v})$ conditioned on the scene state $\mathbf{v}$. To model the extra dimension of the animation, we follow the practice of previous work [Su et al. 2024], which adds multiple feature planes for the scene vector and spatial position, along with a lightweight MLP to capture the correlations between the scene variables.

$$f_{\mathcal{X}\mathcal{V}}(\mathbf{x}, \mathbf{v}) = \bigoplus_{p \in \{x,y,z\}, v \in \mathbf{v}} P(p, v), \tag{15}$$

$$f_{\mathcal{V}}(\mathbf{v}) = \text{MLP}(\texttt{Freq}(\mathbf{v})), \tag{16}$$

where $P(\cdot,\cdot)$ denotes the feature plane for modeling the correlation between the variables and spatial features, while $\texttt{Freq}(\cdot)$ denotes the frequency encoding [Mildenhall et al. 2021] applied to the scene vector $\mathbf{v}$ before feeding it into the MLP.

These two parts are added to assist with our spatial neural vertex feature encoding, which models the full dynamic radiance field

$L_\theta(\mathbf{x}, \omega_o, \mathbf{v})$ effectively. The use of our neural vertex feature encoding significantly reduces memory usage while capturing the full dynamic radiance field $L_\theta(\mathbf{x}, \omega_o, \mathbf{v})$ effectively.

5 IMPLEMENTATION

System. We implement our algorithm using the Mitsuba3 renderer [Jakob et al. 2022]. The neural vertex features are implemented with PyTorch [Paszke et al. 2019], and the multi-layer perceptron (MLP) is implemented using `tiny-cuda-nn` [Müller 2021]. We train our model and render images on an Nvidia RTX 4090 GPU.

Render integration. To render dynamic scenes with our proposed method, we first ray-intersect the camera rays to obtain the primary shading point, along with the view directions ω_o and geometry information. This data is then fed into the network to obtain the approximated radiance L_θ. In our experiments, we perform offline rendering with 32 rays per pixel for static scenes, and for interactive rendering of dynamic scenes, we use 1 ray per pixel with FXAA [DesLauriers 2021] assistance.

Reflectance Factorization. In practice, the outgoing radiance is highly correlated with the surface's reflectance. As used in [Müller et al. 2020], we decouple the texture and radiance by factorizing the reflectance out of the learning target. This allows the network to focus solely on the texture-independent details, alleviating the burden on the network by freeing it from the need to memorize texture information, as shown in Fig. 9.

Additional Inputs. To enhance the model's ability to capture the radiance, we also provide auxiliary geometry features as additional inputs to the network. Specifically, we incorporate the outgoing direction ω_o, as well as the albedo and surface normal at the hit point $\mathbf{x}$. The albedo is encoded using OneBlob encoding [Müller et al. 2019], while the outgoing direction ω_o and surface normal are encoded with spherical harmonic basis functions.

Architecture & Training. For static and dynamic scenes, we assign 4 and 8 learnable parameters per vertex, respectively. The MLP has 4 hidden layers with 256 neurons each. Training uses Adam with learning rate 1×10^{-3}, decayed by 0.33 every one-third of the total steps. The RHS sample count M (Eq. 13) starts at 32 and doubles every 1/5 of the training steps [Su et al. 2024]. LOD factors are updated three times early in training, limited to at most 50% of the original vertex count (100% for dynamic scenes) and clamped to $[0, 30]$. The probability mixing factor α in Eq. 9 is fixed at 0.5.

Details of Hash Grid. The hash grid for static scenes is configured with eight resolution levels, with the coarsest resolution set to 4 and a scale factor of 2 applied between successive levels. Each voxel is assigned 8 learnable parameters per resolution level. To explore performance across different capacities, we set the hash map sizes to 17, 18, and 19 for comparison. For dynamic scenes, we adopt the same hyperparameter settings as specified in Su et al. [2024].

6 RESULTS

6.1 Static Scenes

We first evaluate our method on the application of pre-trained surface-based neural caching in static scenes, namely Neural Radiosity [Hadadan et al. 2021], where the positional network inputs could be backed with different encoding schemes, e.g., Fourier basis encoding [Mildenhall et al. 2021], and trainable parametric encoding [Müller et al. 2022; Takikawa et al. 2023]. In this evaluation, we conduct qualitative experiments against a multi-resolution hash grid with varying hash table sizes for each level ($2^{17} - 2^{19}$ features). Both methods are trained in $2-3$ hours. As shown in Fig. 1, Fig. 8 and Tab. 1, grid encoding tends to exhibit blurriness around fine details as the capacity of the hash table decreases. Moreover, the grid-like artifacts could be seen at the smooth transitions of shaded surfaces, which we attribute to the increased hashing collisions over the smaller hash table. On the other hand, our method achieves more accurate modeling of view-dependent surface shading by allocating spatial features in a more coherent manner with scene geometries, while explicitly accounting for the shading variations introduced by the geometric discontinuities along the vertex edges.

6.2 Dynamic Scenes

We then evaluate our method on the dynamic cases previously established by Su et al. [2024]. We test the effectiveness of our encoding in dynamic scenes featuring different types of animated components and varying geometric complexities, including:

Table 1. Our approach vs. multi-resolution hash grid with varying hash table sizes for each level ($2^{17} - 2^{19}$ features). Memory usage (MB) and performance (ms) in the encoding and MLP components in the static scene are reported. Our method's vertex encoding consumes less memory than competing approaches, while also delivering a modest performance gain.

Memory / Time		Dining Room	Living Room	Veach Door
Hash17	Encoding	13.9 / 32.2	13.9 / 32.2	13. 9 / 32.1
	MLP	1.1 / 27.2	1.1 / 27.4	1.1 / 27.4
Hash18	Encoding	33.9 / 32.1	33.9 / 32.1	33.9 / 32.1
	MLP	1.1 / 27.7	1.1 / 27.5	1.1 / 27.4
Hash19	Encoding	41.9 / 32.1	41.9 / 32.2	41.9 / 32.1
	MLP	1.1 / 27.5	1.1 / 27.4	1.1 / 27.4
Ours	Encoding	3.7 / 27.6	4.3 / 23.1	3.8 / 13.6
	MLP	1.1 / 27.1	1.1 / 27.1	1.1 / 23.1

Table 2. Our approach vs. DNR [Su et al. 2024] in the dynamic scenes. We report memory usage (MB) and performance (ms) statistics for the spatial encoding, extra encoding, and MLP components. Our method's vertex encoding achieves substantially greater memory savings, even surpassing the results in the static scene, and delivers a modest performance improvement.

Memory / Time		Dining Room	Living Room	Veach Door
DNR	Encoding	105 / 31.1	105 / 31.1	105 / 31.1
	$f_{\mathcal{X}\mathcal{V}}$	14.0 / 16.0	10.0 / 10.2	12.0 / 13.4
	$f_{\mathcal{V}}$	0.3 / 1.8	0.3 / 1.6	0.3 / 1.7
	MLP	1.2 / 12.9	1.2 / 12.0	1.1 / 12.5
Ours	Encoding	7.4 / 26.4	6.8 / 23.2	17.3 / 12.8
	$f_{\mathcal{X}\mathcal{V}}$	14.0 / 16.1	10.0 / 10.2	12.0 / 13.3
	$f_{\mathcal{V}}$	0.3 / 1.8	0.3 / 1.6	0.3 / 1.7
	MLP	1.1 / 11.8	1.1 / 11.1	1.1 / 11.4

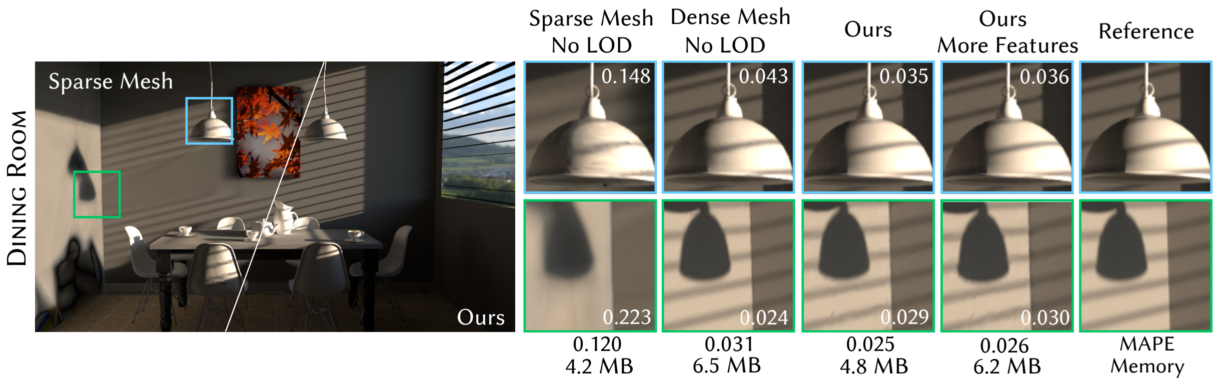

Fig. 5. Varying strategies for capturing fine-grained details in the Dining Room scene. Sparse mesh with no LOD: Poorly represented high-frequency shadow details on the walls due to low vertex density. Dense mesh with no LOD: More detail but with higher memory consumption. Our method: Better results with lower memory consumption via adaptive LOD. Our method with more features: similar quality but with more memory usage.

- The 7D Dining Room scene features rotational environment lighting, two dynamic ceiling lights, a wall with varying reflectance, and three animated objects.
- The 5D Living Room scene includes window blinds for controlling incoming light, a ceiling light with adjustable intensity and height, and an animated table that moves along the xy plane.
- The 6D Veach Door scene consists of three animated meshes, an animated door with varying opening amplitude, and a floor surface with changing reflectance.

We train our model for ~10 hours per scene and compare our approach with equal-time path tracing (PT) and the state-of-the-art neural rendering method for dynamic scenes, Dynamic Neural Radiosity (DNR) [Su et al. 2024]. As shown in Fig. 10, our method achieves visually comparable results with DNR but with far less memory consumption. Specifically, the original grid-based encoding used by Su et al. [2024] tends to blur high-frequency details and exhibits grid-like artifacts. This is due to the fact that our formulation is object-oriented, which allows for efficient modeling of the transform-invariant features of animated meshes. Additionally, our method achieves a roughly 5× reduction of memory footprint with less inference overhead, with model statistics shown in Tab. 2. Additional demonstrations of the animated scene experiments can be inspected in the accompanied video.

6.3 Evaluation

Visualization of Adaptive LOD. We visualize the adaptive LOD results in the static scene rendering experiment, as shown in Fig. 11. The results show that most multi-resolution refinement occurs in regions with low initial mesh density, such as the floor and walls, where additional detail is needed for accurate reconstruction. This demonstrates that our method effectively identifies under-represented areas and adaptively allocates virtual vertices, enabling improved feature representation without unnecessary refinement in well-resolved regions.

Ablation Study on Multi-resolution. We evaluate the effectiveness of our multi-resolution strategy through a comparison of various mesh resolutions with/without LOD. As shown in Fig. 5, the Dining Room scene contains high-frequency shadow details on the walls, which are poorly represented due to the low vertex density in the original mesh. Without LOD, these regions lack sufficient feature support, resulting in noticeable loss of detail. Increasing mesh density through manual subdivision can improve detail, but is resource-intensive. In contrast, our method leverages an adaptive multi-resolution strategy to achieve superior detail preservation using the original sparse mesh as input, while requiring less memory. This highlights the importance of adaptive LOD in capturing fine-grained appearance in under-resolved areas.

Feature Continuity Across Triangle Boundaries. When adjacent triangles use different LOD levels, features across their shared edges may introduce discontinuities. As shown in Fig. 6, rendering results at different training steps reveal this effect. At early stages (steps 200 and 500), visible seams appear along the boundaries. However, as training progresses (steps 5,000 and 10,000), the seams become less noticeable. By the final step (40,000), the discontinuity is no longer perceptible. This demonstrates that, given sufficient training, the network effectively smooths out the discontinuities, producing a seamless feature representation across triangle boundaries, even with different LOD levels.

Neural Path Guiding. We also validate the effectiveness of our method on another family of techniques leveraging neural caches to facilitate rendering, namely neural path guiding [Dong et al. 2023; Huang et al. 2024]. This technique re-parameterizes the network outputs to analytic models (e.g., mixtures of spherical Gaussians) for efficient guided sampling and variance reduction. In this evaluation, we implement their method in Mitsuba3 renderer using PyTorch and apply our technique to it. Following the configuration of Dong et al. [2023], we let the network predict mixtures of normalized spherical Gaussians with 8 lobes. The learned mixtures are trained online and used to do multiple importance sampling (MIS) [Veach and Guibas 1995] with original BSDF sampling at each shading point in unidirectional path tracing. Since the path guiding task involves only a single-view rendering and requires less training time, we use a pre-subdivided mesh to provide sufficient feature resolution. As

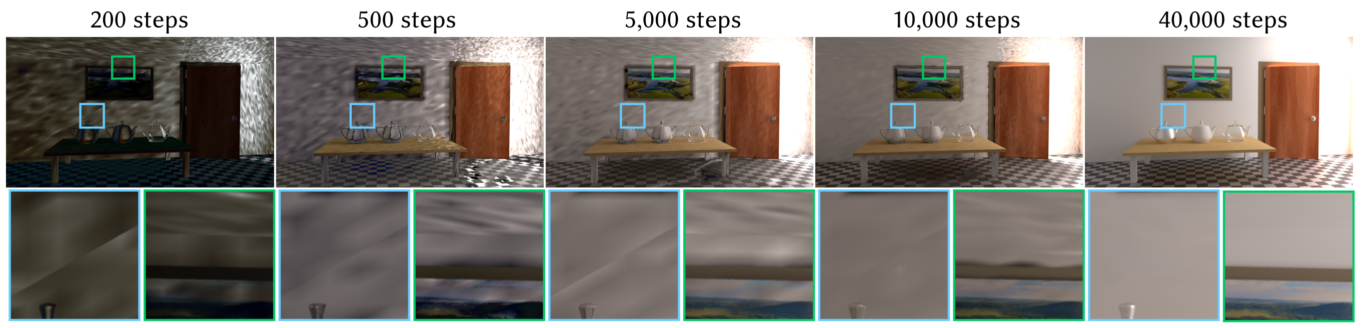

Fig. 6. Discontinuities at triangle boundaries caused by different LOD levels gradually vanish as training progresses. Our neural network–based approach effectively smooths these discontinuities, yielding a seamless feature representation across triangle boundaries, even when different LOD levels are used.

shown in Fig. 7, our method achieves comparable quality with the hashed grid approaches while using much less memory footprint.

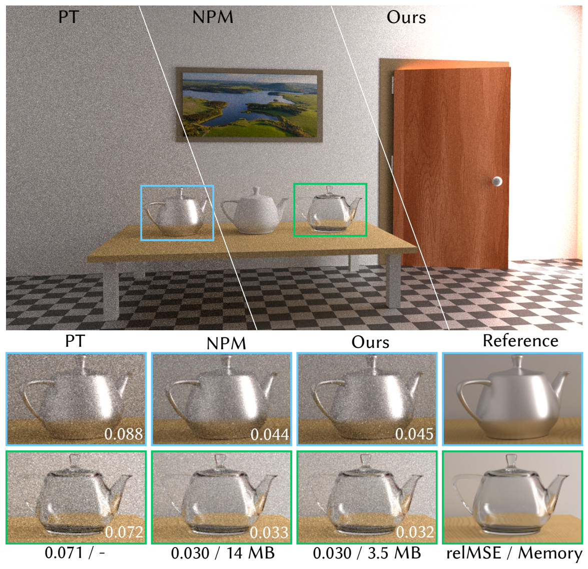

Fig. 7. We validate our approach on the neural path guiding techniques [Dong et al. 2023]. Our method achieves comparable quality with the hashed grid approaches while using much less memory footprint.

7 CONCLUSION, LIMITATIONS, AND FUTURE WORK

In this work, we propose a trainable latent encoding for efficient neural rendering. By encoding spatial information directly at the vertices of the mesh, we achieved more compactness by exploiting the geometric priors, thereby reducing memory overhead compared to traditional 3D-grid-based encoding. Our method captures important surface details while maintaining computational efficiency. We boost low-density meshes with a virtual multi-resolution scheme that refines features where training loss is high, enabling recovery of high-frequency lighting in undersampled areas. Our method achieves significant memory savings and comparable or better image quality compared to state-of-the-art methods. Furthermore, our vertex feature formulation has the potential to perform various neural rendering tasks.

Despite the effectiveness of our method, there are still some limitations. Since our approach encodes features directly at mesh vertices, it is inherently restricted to surface-based representations and cannot be directly applied to volumetric data or scenes without explicit surface geometry. Additionally, our method focuses on subdividing features at the vertices without simplifying the underlying mesh structure. This design choice was made to preserve geometric detail, as mesh simplification may result in a loss of details. However, this could be explored as a future improvement for more efficient representations while retaining high expressiveness.

In dynamic scenes from the video, we still observe subtle artifacts and frame-to-frame inconsistencies, whereas these issues are absent in static scenes. This is due to the dynamic scene's high dimensionality and the reliance on grid-based encodings ($f_{\mathcal{X}\mathcal{V}}$ and $f_{\mathcal{V}}$) for modeling the dynamic components. More broadly, using a shallow MLP to model such components in a high-dimensional space can induce temporal instabilities (the same issue with DNR). In particular, when moving objects cast shadows beneath meshes, the resulting regions contain high-frequency radiance that the network struggles to capture. This will be one of our future works to tackle this issue.

Moreover, we also plan to explore hybrid representations that combine surface-based encoding with sparse volumetric features to support more diverse scene types. Furthermore, we aim to investigate methods for mesh simplification in conjunction with feature refinement, allowing for more efficient representation while maintaining high expressiveness. We also note that our method could be straightforwardly applied to alternative geometry primitives, such as quadrilaterals, and we leave the validation of its practicality and effectiveness for future exploration.

ACKNOWLEDGMENTS

We thank the following creators of the test scenes: Wig42 (Dining Room, Living Room), Jaakko Lehtinen(Veach Door), Jay-Artist (White Room), as well as the efforts for converting scene formats by Benedikt Bitterli [2016]. This work is supported by the National Key R&D Program of China (No.2023YFF0905103), NSFC of China (No.62172013), and Shenzhen Science and Technology Program (KJZD20230923114114028).

REFERENCES

Bao and Yang, Zeng Junyi, Bao Hujun, Zhang Yinda, Cui Zhaopeng, and Zhang Guofeng. 2022. NeuMesh: Learning Disentangled Neural Mesh-based Implicit Field for Geometry and Texture Editing. In *European Conference on Computer Vision (ECCV)*.

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Ang Cao and Justin Johnson. 2023. Hexplane: A fast representation for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 130–141.

Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. 2020. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16*. Springer, 608–625.

Arno Coomans, Edoardo A Dominci, Christian Döring, Joerg H Mueller, Jozef Hladky, and Markus Steinberger. 2024. Real-time Neural Rendering of Dynamic Light Fields. In *Computer Graphics Forum*, Vol. 43. Wiley Online Library, e15014.

Matt DesLauriers. 2021. *glsl-fxaa* . https://github.com/mattdesl/glsl-fxaa

Stavros Diolatzis, Julien Philip, and George Drettakis. 2022. Active exploration for neural global illumination of variable scenes. *ACM Transactions on Graphics (TOG)* 41, 5 (2022), 1–18.

Honghao Dong, Guoping Wang, and Sheng Li. 2023. Neural Parametric Mixtures for Path Guiding. In *ACM SIGGRAPH 2023 Conference Proceedings*. 1–10.

Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. 2023. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12479–12488.

Jonathan Granskog, Fabrice Rousselle, Marios Papas, and Jan Novák. 2020. Compositional neural scene representations for shading inference. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 135–1.

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural radiosity. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–11.

Jiawei Huang, Akito Iizuka, Hajime Tanaka, Taku Komura, and Yoshifumi Kitamura. 2024. Online Neural Path Guiding with Normalized Anisotropic Spherical Gaussians. *ACM Transactions on Graphics* 43, 3 (2024), 1–18.

Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. 2021. Interactive Monte Carlo denoising using affinity of neural features. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–13.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. 2022. *Mitsuba 3 renderer*. https://mitsuba-renderer.org.

James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (aug 1986), 143–150. https://doi.org/10.1145/15886.15902

Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Liefeng Bo. 2023. Compressing volumetric radiance fields to 1 mb. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4222–4231.

M Mahajan, F Hofherr, and D Cremers. 2024. MeshFeat: Multi-Resolution Features for Neural Fields on Meshes. In *European Conference on Computer Vision (ECCV)*. arXiv:2407.13592

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)* 41, 4 (2022), 1–15.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural importance sampling. *ACM Transactions on Graphics (ToG)* 38, 5 (2019), 1–19.

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural control variates. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–19.

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–16.

Thomas Müller. 2021. *tiny-cuda-nn*. https://github.com/NVlabs/tiny-cuda-nn

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 523–540.

Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. 2023. Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16632–16642.

Rui Su, Honghao Dong, Jierui Ren, Haojie Jin, Yisong Chen, Guoping Wang, and Sheng Li. 2024. Dynamic Neural Radiosity with Multi-grid Decomposition. In *SIGGRAPH Asia 2024 Conference Papers*. 1–12.

Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. 2022. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–9.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11358–11367.

Towaki Takikawa, Thomas Müller, Merlin Nimier-David, Alex Evans, Sanja Fidler, Alec Jacobson, and Alexander Keller. 2023. Compact Neural Graphics Primitives with Learned Hash Probing. In *SIGGRAPH Asia 2023 Conference Papers*.

Eric Veach and Leonidas J Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 419–428.

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.

Songyin Wu, Sungye Kim, Zheng Zeng, Deepak Vembar, Sangeeta Jha, Anton Kaplanyan, and Ling-Qi Yan. 2023. ExtraSS: A Framework for Joint Spatial Super Sampling and Frame Extrapolation. In *SIGGRAPH Asia 2023 Conference Papers*. 1–11.

Peilin Zhao and Tong Zhang. 2015. Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*. PMLR, 1–9.

Chuankun Zheng, Yuchi Huo, Hongxiang Huang, Hongtao Sheng, Junrong Huang, Rui Tang, Hao Zhu, Rui Wang, and Hujun Bao. 2024. Neural Global Illumination via Superposed Deformable Feature Fields. In *SIGGRAPH Asia 2024 Conference Papers*. 1–11.

Chuankun Zheng, Yuchi Huo, Shaohua Mo, Zhihua Zhong, Zhizhen Wu, Wei Hua, Rui Wang, and Hujun Bao. 2023. NeLT: Object-Oriented Neural Light Transfer. *ACM Trans. Graph.* 42, 5, Article 163 (aug 2023), 16 pages. https://doi.org/10.1145/3596491

Zhihua Zhong, Jingsen Zhu, Yuxin Dai, Chuankun Zheng, Guanlin Chen, Yuchi Huo, Hujun Bao, and Rui Wang. 2023. FuseSR: Super Resolution for Real-time Rendering through Efficient Multi-resolution Fusion. In *SIGGRAPH Asia 2023 Conference Papers*. 1–10.

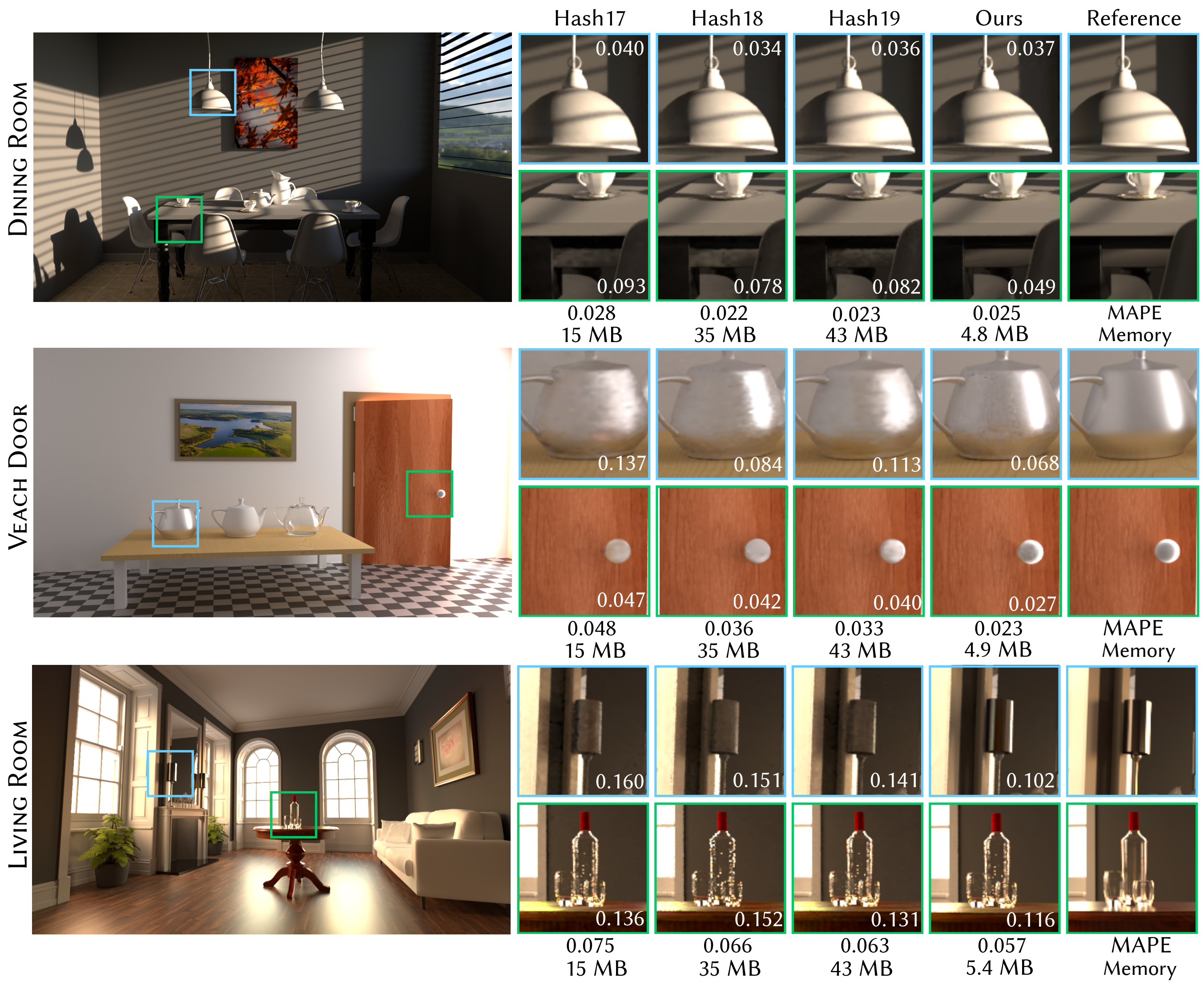

Fig. 8. Evaluating the effectiveness of neural vertex feature on Neural Radiosity [Hadadan et al. 2021]. We compare the visual quality of our method against multi-resolution hash encoding [Müller et al. 2022] with different hash table sizes (2^{17}-2^{19} features for each level) in 3 static scenes. However, it tends to exhibit more blurriness on high-frequency details and grid-like artifacts as the feature capacity decreases. Our method achieves much less memory consumption at comparable visual quality by further culling the features from the empty scene spaces.

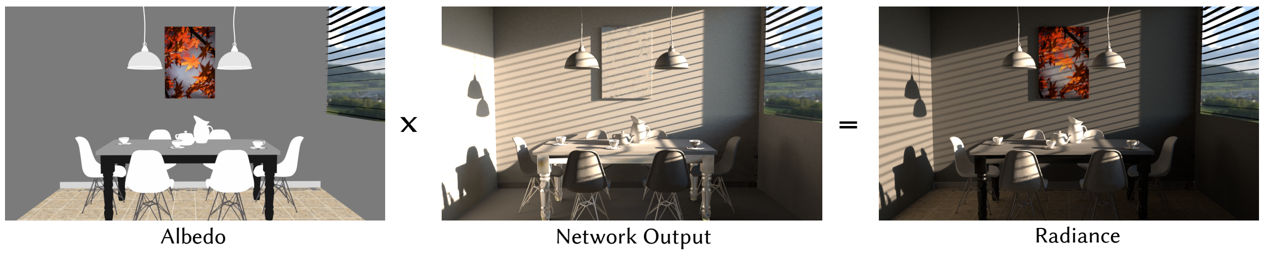

Fig. 9. Visualization of reflectance factorization. We let the network predict the direction-independent irradiance term by factorizing out the reflectance term. This technique helps to alleviate the burden of the network while helping recover the high-frequency details of the textures.

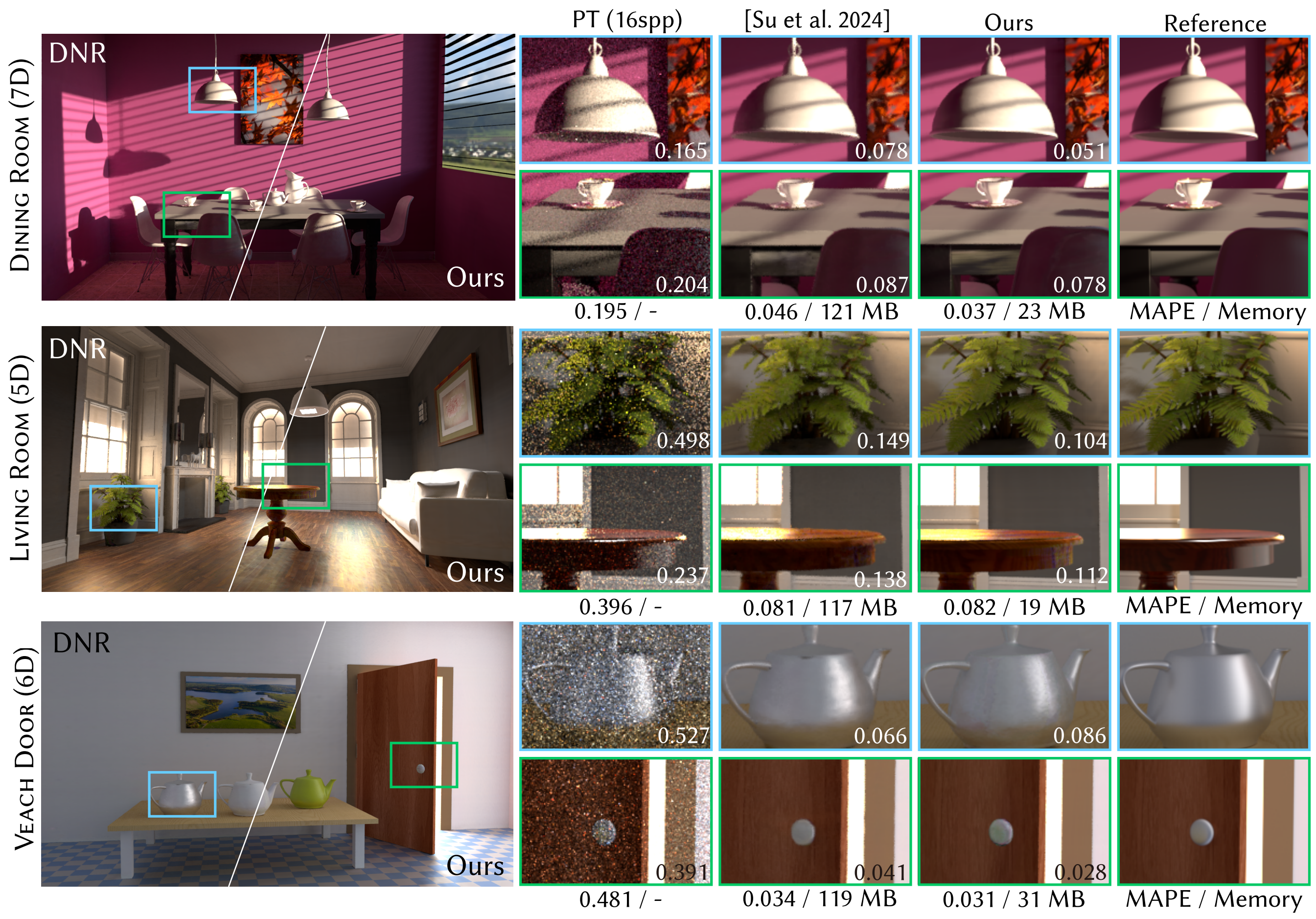

Fig. 10. Evaluating the effectiveness in three dynamic scenes featuring various animated components. Neural vertex features substitute the 3D hash grid of Su et al. [2024] to encode purely spatial information, cutting memory and avoiding the grid-like artifacts that hashing's random interpolation introduces in smooth regions while also relieving the network from handling hash collisions. The dynamic component remains represented by the original 2D feature planes and small MLPs. The surface-aligned locality of vertex features reproduces gradual shading transitions in both diffuse and specular areas. Please refer to the supplemental video for frame-by-frame comparisons.

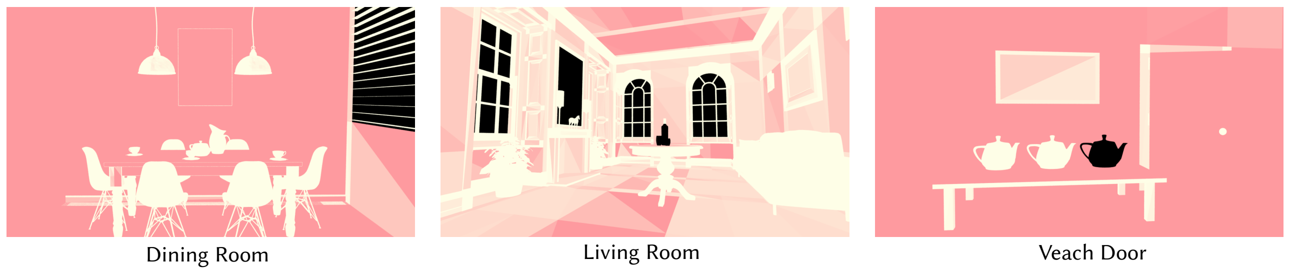

Fig. 11. Visualization of the learned LOD factors (relatively higher subdivision levels are shown in warmer colors). Refinement is concentrated in regions with low initial mesh density, such as floors and walls, where finer detail is required. This highlights the effectiveness of our method in adaptively allocating virtual vertices to under-represented areas, while avoiding unnecessary refinement in well-resolved regions.