Gaussian Compression for Precomputed Indirect Illumination

ZHI ZHOU, Tencent, China and University of Science and Technology of China, China
CHAO LI, Tencent, China
ZHENYUAN ZHANG, Tencent, China
MINGCONG TANG, Tencent, China
ZIBIN LI, Tencent, China
SHUHANG LUAN, Tencent, China
ZHANGJIN HUANG, University of Science and Technology of China, China

Fig. 1. Performance comparison in Havana test scene. The first two images show the lighting results obtained from recovered light probe data, which is compressed using different methods. To facilitate a clearer comparison, textures are removed to highlight the shading results. We use bits per probe (bpp) and compression ratio to measure different methods. The third image shows the render result with the original probe data. The last image presents the complete rendered image from this viewpoint. Compared to block-wise PCA (BPCA) [21], our method, Gaussian Probe Compression (GPC), demonstrates significant advantages in both compression ratio and image quality. Notably, our approach effectively restores the color bleeding effects present in the scene.

Precomputed global illumination (GI) techniques, such as light probes, particularly focus on capturing indirect illumination and have gained widespread adoption. However, as the scale of the scenes continues to expand, the demand for storage space and runtime memory for light probes also increases substantially. To address this issue, we propose a novel Gaussian fitting compression technique specifically designed for light field probes, which enables the use of denser samples to describe illumination in complex scenes. The core idea of our method is utilizing low-bit adaptive Gaussian functions to store the latent representation of light probes, enabling parallel and high-speed decompression on the GPU. Additionally, we implement a custom gradient propagation process to replace conventional inference frameworks, like PyTorch, ensuring an exceptional compression speed.

At the same time, by constructing a cascaded light field texture in real-time, we avoid the need for baking and storing a large number of redundant light field probes arranged in the form of 3D textures. This approach allows us to achieve further compression of the memory while maintaining high visual quality and rendering speed. Compared to traditional methods based on Principal Component Analysis (PCA), our approach consistently yields superb results across various test scenarios, achieving compression ratios of up to 1:50.

CCS Concepts: • **Computing methodologies** → **Rendering**; *Image compression.*

ACM Reference Format:
Zhi Zhou, Chao Li, Zhenyuan Zhang, Mingcong Tang, Zibin Li, Shuhang Luan, and Zhangjin Huang. 2025. Gaussian Compression for Precomputed Indirect Illumination. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25), August 10–14, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3721238.3730758

Authors' Contact Information: Zhi Zhou, doji@mail.ustc.edu.cn, Tencent, Hangzhou, China and University of Science and Technology of China, Hefei, China; Chao Li, superli@zju.edu.cn, Tencent, Hangzhou, China; Zhenyuan Zhang, cryscan@umich.edu, Tencent, Shanghai, China; Mingcong Tang, 2301210400@stu.pku.edu.cn, Tencent, Shanghai, China; Zibin Li, 22135085@zju.edu.cn, Tencent, Shanghai, China; Shuhang Luan, luanshuhang0203@163.com, Tencent, Hangzhou, China; Zhangjin Huang, zhuang@ustc.edu.cn, University of Science and Technology of China, Hefei, China.

SIGGRAPH Conference Papers '25, August 10–14, 2025, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1540-2/2025/08
https://doi.org/10.1145/3721238.3730758

1 Introduction

In recent years, advances in real-time rendering technology have significantly narrowed the gap between video games and film productions. New technologies such as Lumen, Nanite [27], real-time

path tracing [22, 34], and AI denoising [1, 5] have produced stunning visual effects. However, they require robust hardware support. In devices with limited computational power, achieving ideal global illumination (GI) effects requires reliance on a series of precomputed techniques to bake scene lighting in advance, such as lightmaps and light probes [17]. As scenes become increasingly expansive, the corresponding precomputed data also grows, placing greater demands on device storage and memory.

While there has been considerable research on texture compression, there are relatively few methods addressing the compression of light probes, which represent the distribution of light fields in 3D space. The challenge lies in the irregular arrangement of probes, which are often scattered only on the surface of the scene and do not necessarily conform to the arrangement of 3D textures, making it difficult to apply conventional texture compression techniques.

To address this issue, we propose an adaptive compression method based on Gaussian fitting to compress high-dimensional signals in the space. This approach utilizes freely distributed Gaussian functions to fit the probe information within the scene. The parameters of the Gaussian functions can be updated using gradient descent, allowing for the automatic identification of the most suitable arrangement for the current scene. For a given probe to be compressed, we compute the weighted sum of all Gaussian functions influencing that probe as a predicted value for its latent feature, which is then decoded by a lightweight multi-layer perceptron (MLP) to obtain the decompressed high-dimensional lighting data for the probe. The entire optimization process is end-to-end, allowing us to store only the network model and the sparse, quantized Gaussian parameters, rather than the high-dimensional lighting information for each probe, thereby significantly reducing storage space.

Furthermore, to accelerate the entire compression process, we implement a custom Gaussian function inference and backpropagation process using CUDA, achieving the desired compression results in a short time. This method takes into account the spatial locality of each Gaussian function and uses only adjacent Gaussian functions for forward inference on each probe. The strategy also reduces the gradient computation during the backward process. Compared to general inference frameworks like PyTorch, our implementation significantly speeds up the overall compression workflow.

In addition to reducing the storage space required for light probes, we also focus on minimizing their runtime memory usage. To perform per-pixel shading using precomputed light probes, it is typically necessary to construct a 3D texture filled with light information for real-time interpolation, such as the volumetric lightmap (VLM) in Unreal Engine [7] or the adaptive probe volumes (APV) [28] in Unity Engine. However, these approaches involve placing dense 3D textures in space, treating each texel as a light probe. The dense probe distribution not only increases the precomputation cost but also consumes substantial memory. To address this issue, we propose cascaded lighting volume (CLV), a method that is able to interpolate relatively sparse probe data into 3D textures and only compute detailed volumetric lightmap surrounding the current viewport. Our experiments show that CLV achieves a balance between shading quality and memory usage. To sum up, our main contributions are as follows:

- A novel approach for compressing precomputed light field probes that exploits spatial relevance through Gaussian functions and yields superb performance in both compression and restoration.
- An efficient implementation of our optimization-based compression pipeline, enabling fast convergence in large-scale scenes.
- A novel probe interpolation strategy that achieves a balance between rendering quality and runtime memory consumption.

2 Related Work

2.1 Probe Based Global Illumination

Light probe is a crucial method for GI and can be broadly categorized into two types: precomputed light probes and real-time updated light probes. Light field probe [17] computes the shading of glossy and specular materials by storing precomputed radiance, normals, and distance maps at each probe, effectively serving as an improvement over irradiance caching [14]. Subsequent works [30] further optimize the placement and tracing methods for these probes.

Another category of light field probes employs spherical harmonic (SH) functions instead of texture maps to represent the light field information for each probe and is our main compression target. This method shares similarity with the irradiance volume [8] and is primarily based on the Precomputed Radiance Transfer (PRT) [26] framework, in which the probe represents the light field by SH coefficients and is primarily used for shading diffuse materials. During runtime, each pixel approximates its surface light field by interpolating information from surrounding probes. This method incurs minimal storage and computational overhead, making it widely adopted in mobile devices. Various probe placement techniques such as tetrahedral tessellation [6], surface tiling, and 3D grid can be used to fit the scene geometry. Our compression scheme can adaptively adjust the distribution of Gaussian functions according to the probe distribution, thus imposing no restrictions on the probe placement methods.

Real-time light probes primarily rely on hardware-accelerated ray tracing to gather light field information, as seen in techniques like Dynamic Diffuse Global Illumination (DDGI) [16]. This approach supports dynamic lighting, with each probe storing illumination information in an octahedral mapping manner during runtime, while depth information is retained to mitigate light leaking issues. However, this method has substantial hardware requirements, making it challenging to implement broadly. In contrast to previous methods, the screen space probe in Lumen [27] attaches probes to the surfaces of objects corresponding to screen space pixels and collects dynamic lighting for these probes in real-time, thereby reducing the light leaking phenomena associated with traditional probe schemes.

2.2 Light Field Compression

Our compression target is the light field SH coefficients associated with each probe, which can be abstracted as high-dimensional signals in the spatial domain. To achieve this target, a series of dimensionality reduction methods utilizing Principal Component Analysis [3] (PCA) have been proposed. The simplest global PCA method is insufficient for handling complex scenes with low inter-region correlations. Blockwise PCA (BPCA) [21] attempts to partition the

points to be compressed into clusters and perform local PCA. Cluster PCA (CPCA) [25] first segments the signals into clusters using a modified K-Means [15] algorithm, followed by PCA within each cluster. While these methods achieve more accurate results locally due to the partition strategy, they suffer from discontinuities between clusters. Overlapping clusters in WBPCA [21] can alleviate this issue, but do not completely resolve it. Moving Basis Decomposition (MBD) [24] addresses this problem by decomposing the signal into sparse high-dimensional vectors and dense low-dimensional coefficient grids, which can be viewed as a smoother variant of BPCA. However, the compression rate is not particularly high, and the grid structure employed is challenging to adapt for probes with non-fixed placements.

While our work specifically targets compressed radiance representation for real-time rendering, prior research on radiance caching has explored alternative approaches to efficient radiance encoding. Classical methods such as Photon Mapping [10] directly store incident radiance by caching photon hits on surfaces, enabling efficient Monte Carlo integration of the rendering equation. More recent data-driven techniques further optimize this paradigm: Radiance Regression Functions [23] employ neural networks to compactly encode precomputed illumination, leveraging spatial partitioning (e.g., kd-trees [2]) to balance accuracy and computational cost. Neural Radiance Fields [18] employs MLP to model the radiance and enables novel view synthesis for both synthetic and real-world scenes, while Neural Radiance Caching (NRC) [20] combines real-time neural approximation with physical light transport to handle complex multi-bounce indirect lighting.

Although these methods primarily address challenges in real-time/offline rendering or novel-view synthesis, their underlying representations (e.g., neural embeddings, spatial hierarchies) share theoretical connections with radiance compression. Crucially, our approach diverges by introducing a hybrid Gaussian-neural representation that explicitly optimizes for compression ratio and decoding speed and exploiting inter-probe correlations through local Gaussian function, which is a direction underexplored in caching-focused works.

3 Method

3.1 Compression Framework

Our compression objective focuses on the light field probe. Depending on the GI method employed, the data associated with the probes can take various forms. In the context of our study, we compute the projection value L^i of the indirect lighting onto a specific SH basis function at each probe using Eq. (1), which serves as the coefficient for that basis function. The probe data consists of n SH coefficients (depending on the order of the SH used) and form a vector $\mathbf{L}$.

$$L^i = \int L_{indirect}(\mathbf{s}) Y^i(\mathbf{s}) \mathrm{d}\mathbf{s}, \ i = 1, 2, \ldots, n, \tag{1}$$

where $L_{indirect}$ represents the sampled value of incident indirect radiance at the current position in the direction $\mathbf{s}$, while Y^i denotes the basis function value of the i-th SH basis function. Detailed explanation of our rendering method can be found in appendix.

Regardless of the GI method employed, we can abstract the light probe into a set of discrete data pairs (x_i, y_i). Here, $x_i \in \mathbb{R}^3$ represents the position of the probe i, while $y_i \in \mathbb{R}^D$ denotes the corresponding probe information, D denotes the dimension of light probe information (for instance, the number of SH coefficients, typically 3, 12, or 27). The baking of the probe can be viewed as a function $f : \mathbb{R}^3 \to \mathbb{R}^D$, such that $y_i = f(x_i)$. Consequently, our research objective can be reformulated as approximating $f(x)$ with a function $f'(x)$ that utilizes fewer parameters compared to the original data. The goal is to minimize the error $||\mathbf{y} - f'(\mathbf{x})||, \mathbf{y} = \{y_i\}, \mathbf{x} = \{x_i\}$, thereby achieving data compression.

To solve this issue, we propose the use of a set of Gaussian functions to fit the latent feature distribution of the probes. The latent feature is decoded through a lightweight neural network to obtain the SH vector for the probe. Our method leverages GPU to achieve high-speed compression and decompression, allowing for adaptive adjustment of the Gaussians based on the probe distribution. This approach not only achieves a high compression rate but also minimizes information loss.

3.2 Gradient Descent Gaussian Fitting

The framework of our compression pipeline is shown in Figure 2. The core of our method involves using a specified number of 3D Gaussian functions to fit the latent feature of probe data that is freely distributed in 3D space. Each Gaussian function carries a D-dim latent vector for the following decoding.

For optimization convenience, we adopt the same definition as the 3D Gaussian Splatting (3DGS) [12] technique to represent the Gaussian functions. A Gaussian function G centered at $\mu \in \mathbb{R}^3$ is defined as follows, with its covariance matrix decomposed into two matrices: scale S and rotation R.

$$G(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)) \tag{2}$$

$$\Sigma = RSS^T R^T \tag{3}$$

Concretely, for Gaussian function G_j, we store its 3D scale vector $s_j \in \mathbb{R}^3$, quaternion 4D rotation vector $q_j \in \mathbb{R}^4$, 3D position $\mu_j \in \mathbb{R}^3$ and latent code $F_j \in \mathbb{R}^D$. Assuming we use K Gaussian functions to compress the probe data, for any given position $\mathbf{p} \in \mathbb{R}^3$, we can identify all the Gaussians that cover this point and sum their contributions to obtain the latent vector $F(\mathbf{p})$:

$$F(\mathbf{p}) = (F^0(\mathbf{p}), F^1(\mathbf{p}), \ldots, F^{n-1}(\mathbf{p})) \tag{4}$$

$$F^i(\mathbf{p}) = \sum_{j}^{\mathbf{p} \in R(G_j)} F_j^i G_j(\mathbf{p}), \ i = 0, 1, \ldots, n-1 \tag{5}$$

$$R(G_j) = \{\mathbf{p} \mid |\mathbf{p} - \mu_j| < 3 \max(s_j^0, s_j^1, s_j^2)\} \tag{6}$$

Here, $F^i(\mathbf{p})$ represents the estimated value of the i-th channel at position $\mathbf{p}$ using the Gaussian functions. $R(G_j)$ denotes the influence range of Gaussian j. We estimate this range by taking the bounding sphere formed by the maximum scale, which encompasses the region with a confidence level of 99%. While this representation is not optimal and may include redundant cubes, the training and inference

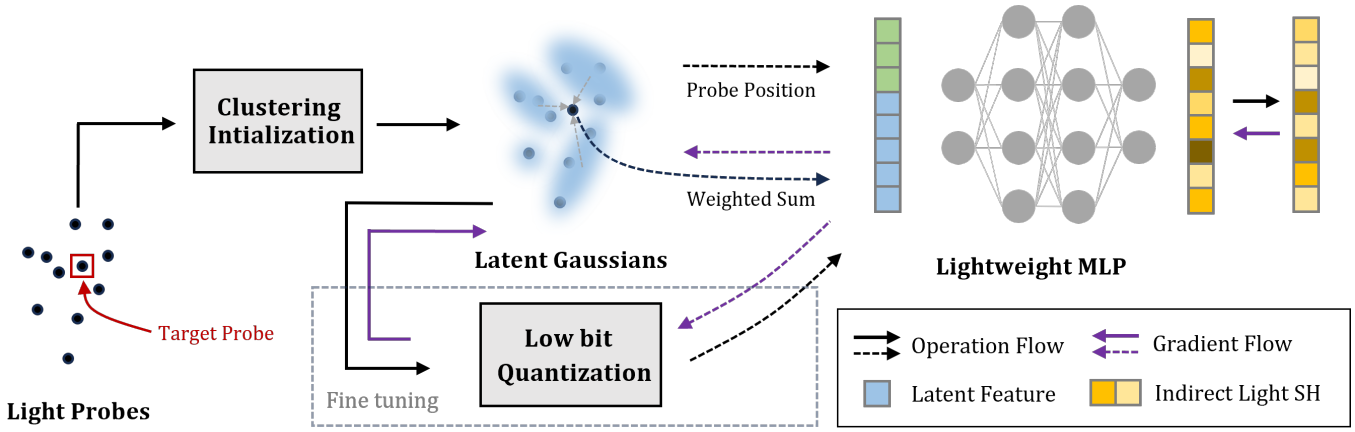

Fig. 2. Overview of our compression pipeline. Given the light probes scattered throughout the scene, we first employ a clustering algorithm to initialize Gaussian parameters. For a target probe, we identify all Gaussians that cover it and compute a latent feature for this probe through a weighted sum. This latent feature, along with the probe's position, is input into a lightweight MLP to predict the SH vector. We jointly update both the Gaussian and the network during this process. Once the training converges, we quantize the Gaussian parameters to fixed low bit integers and perform fine-tuning on the model. Finally, we freeze the Gaussian parameters and continue training the decoder network to adapt to the quantized discrete values.

times are acceptable in our experiments. After obtaining the latent feature $F(\mathbf{p})$ of probe $\mathbf{p}$, we concatenate $F(\mathbf{p})$ with position $\mathbf{p}$ and use a two-layer MLP Φ with parameter θ to decode the combination vector and get the final estimated SH vector $\hat{L}(\mathbf{p})$:

$$\hat{L}(\mathbf{p}) = \Phi([F(\mathbf{p}), \mathbf{p}]; \theta) \tag{7}$$

Suppose the probe number is N, the ground truth at position $\mathbf{p}_k$ is $L(\mathbf{p}_k)$, our objective is to fit the data of all probes using K Gaussian kernels while minimizing the mean squared error of the reconstruction. The loss $\mathcal{L}(G, \theta)$ can be calculated as follows:

$$\mathcal{L}(G, \theta) = \frac{1}{N} \sum_{k=1}^{N} ||L(\mathbf{p}_k) - \hat{L}(\mathbf{p}_k)||^2, \tag{8}$$

$$G = \{G_0, \ldots, G_{M-1}\}, G_j = \{\mu_j, s_j, q_j, F_j\} \tag{9}$$

This formulation allows us to quantify the discrepancy between the actual SH coefficients and those predicted by our compression model. The optimization target can be stated as follows:

$$\arg\min_{G,\theta} \mathcal{L}(G, \theta) \tag{10}$$

During the initialization, we first perform K-Means [15] clustering on the positions of the input probes to obtain K cluster centers, which serve as the initial positions for the Gaussian functions. The scale of each function is initialized based on the minimum distance between the cluster centers, while the initial latent feature for each Gaussian function is set as a random vector. Gradient descent is applied afterwards to update the parameters of each Gaussian and MLP network until convergence.

Forward pass. During the forward phase, we utilize K Gaussians to compute the latent feature for each probe by Eq. (5). To accelerate this process, we implement custom CUDA kernel that allows for the parallel computation of the function values at each target location. Inspired by 3DGS [12], we partition the scene space into uniform cubes. For each Gaussian function, as illustrated in Figure 3, we invoke a thread to compute its maximum radius and record the cubes occluded by the corresponding sphere of influence. For each intersection between the sphere and a cube, we record the cube ID as an identifier for that intersection. By aggregating all intersection IDs and utilizing GPU sorting, we can determine the Gaussians associated with each cube. Finally, we perform parallel computations for all probes. For each probe, we first identify the cube it resides in and use the Gaussians that associate with this cube to calculate the predicted latent feature. This approach significantly enhances the efficiency of the inference process, allowing for rapid evaluations across the entire set of probes.

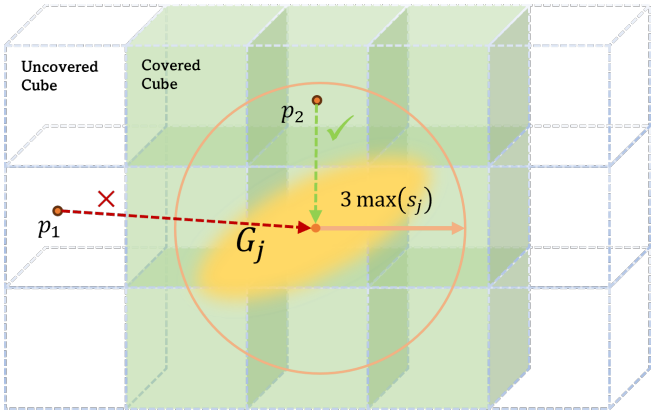

Fig. 3. Our implementation of the forward pass. Green cubes indicate the area influenced by Gaussian G_j. We use the scale parameter to determine the radius of G_j by Eq. (6). For probe p_2 in green area, its latent feature involves G_j while p_1 doesn't.

Backward pass. After obtaining the predicted values for the probes, the loss between the current predictions and the ground truth is

used to calculate the gradients of the Gaussians and the decoding network. Since the forward phase utilized custom CUDA functions for parallel computation, we cannot completely depend on the automatic differentiation mechanism of Pytorch for backpropagation; thus, we derive the gradients manually.

The backward process corresponds to the forward process. We begin with the latent features outputted for each probe during the forward pass and identify all Gaussians that influence each probe. Then the gradients of the features are propagated back to each corresponding attribute of the Gaussians, which include the center position μ, scale s, rotation q, and the latent code F. The gradient for a specific attribute of a Gaussian is computed as the sum of the gradients from all probes that are influenced by this Gaussian. Similar to the forward phase, we implement the parallelized backpropagation using custom CUDA kernel. This approach allows us to efficiently compute the gradients for all Gaussian parameters.

Starting from probe $\mathbf{p}_k$, we can derive the gradient contribution of this single probe to the attributes of function G_j. Assuming that $\frac{\partial \mathcal{L}}{\partial F^i(p_k)}$ is known, which can be obtained directly from the inference framework, the gradient $\frac{\partial \mathcal{L}}{\partial F_j}$ is given by:

$$\frac{\partial \mathcal{L}}{\partial F_j^i} = \sum_{\mathbf{p}_k}^{\mathbf{p}_k \in R(G_j)} \frac{\partial \mathcal{L}}{\partial F^i(\mathbf{p}_k)} \cdot \frac{\partial F^i(\mathbf{p}_k)}{\partial F_j^i} \tag{11}$$

$$= \sum_{\mathbf{p}_k}^{\mathbf{p}_k \in R(G_j)} \frac{\partial \mathcal{L}}{\partial F^i(\mathbf{p}_k)} \cdot G_j(\mathbf{p}_k) \tag{12}$$

Moreover, one can compute the gradients $\frac{\partial \mathcal{L}}{\partial \mu_j}$, $\frac{\partial \mathcal{L}}{\partial s_j}$, $\frac{\partial \mathcal{L}}{\partial q_j}$, using a similar approach. The concrete derivation of these gradients is included in the appendix.

3.3 Adaptive Low-bit Quantization

Upon convergence of the above optimization, we further compress the model by mapping the scale s, rotation q, and latent feature F of the Gaussian model to b-bit integers based on their respective ranges. Taking the scale as an example, for the i-th dimension of the scale of Gaussian G_j, denoted as s_j^i, we extract the parameter range (s_{max}^i, s_{min}^i) for this channel and perform the following quantization:

$$\hat{s}_j^i = \lfloor \frac{s_j^i - s_{min}^i}{s_{max}^i - s_{min}^i} \cdot (2^b - 1) \rceil \tag{13}$$

$$\tilde{s}_j^i = \hat{s}_j^i \cdot \frac{s_{max}^i - s_{min}^i}{2^b - 1} + s_{min}^i \tag{14}$$

After quantization, each Gaussian only needs to store b-bit integer $\hat{s}_j^i$. During inference, these integers are used to compute the restored parameters $\tilde{s}_j^i$. Naturally, this quantization process lead to a loss in compression accuracy. To mitigate this loss, as illustrated in Figure 2, we retain the original Gaussian parameters and fine-tune the entire compression model using the quantized parameters for 10% more steps. Straight-through estimator [33] is used to ensure that gradients can flow back to the underlying continuous Gaussian parameters. When these parameters exceed the quantization range, we apply truncation to prevent any parameters from drifting outside the quantized range.

Following this stage, we completely freeze the quantized Gaussian parameters, discard the original floating-point parameters, and continue training the decoding network for 10% more steps to adapt it to the discrete Gaussian parameters. Finally, we save the quantized low-bit Gaussian parameters along with the network model, resulting in a compressed representation of the original probe data.

3.4 Cascaded Lighting Volume

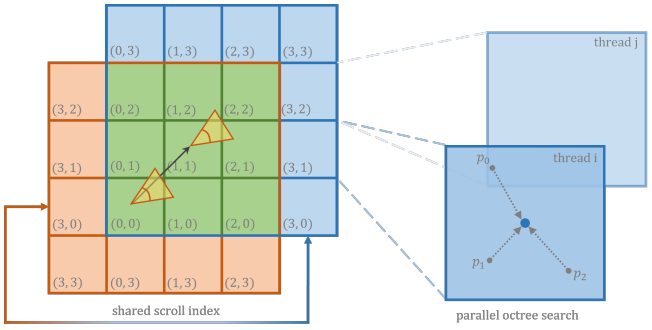

Fig. 4. CLV Framework. In each frame, we compare the current viewpoint's position and direction with last frame to determine if any changes have occurred. Parallel probe search for the corresponding new blocks is performed afterwards. We compute the weighted sum of the probe SH vectors for the center of the block. The new blocks (blue) and the blocks that are about to be discarded (orange) share the same scroll coordinates, thereby minimizing texture updates.

While our method primarily achieves storage space reduction. Additional methods need to be explored to utilize the light field probe for runtime shading. Current industry solutions like VLM in Unreal and APV in Unity are to arrange the probes in 3D textures and after baking, each pixel is able to sample this texture and get the lighting SH vector by trilinear interpolation. Probes are placed with uniform density in both open areas and complex regions, leading to data redundancy and increased memory usage.

To address this issue, we propose Cascaded Lighting Volume (CLV), which allows for the placement of sparse probes based on the scene geometry while supporting per-pixel interpolation at the same time. We maintain an octree structure to store probe data during runtime, along with a multi-level voxel grid (cascaded volumes) centered around the current viewpoint, where each volume corresponds to a different size 3D texture representing indirect lighting information, similar to the Light Propagation Volumes [11]. This method identifies the blocks in the grid that need to be updated for the current frame. For each of these blocks, we search for the probes located within them and compute the lighting at the center of the block. The lighting SH vector is then updated in the 3D texture on the GPU, enabling per-pixel interpolation for shading the objects.

Specifically, each pixel selects the corresponding volume for indirect lighting interpolation based on its distance to the viewpoint. The closer a pixel is to the viewpoint, the smaller the texture used,

resulting in more detailed lighting effects. As the viewpoint moves within the scene, we need to dynamically update the data for these volumes based on the current viewpoint position. For each frame, we first compare the viewpoint position of the current frame with that of the previous frame to determine whether an update to the volume is necessary. If an update is required, we retrieve all blocks that need updating and, for each block, quickly obtain all probes located within that block from the probe octree, along with their corresponding indirect lighting SH vectors. Using the same methodology as training, we first identify the Gaussian functions influencing each probe, then compute per-probe feature values, and finally feed these into the network for inference to reconstruct the decompressed SH coefficients. We then calculate the contribution of each probe to its corresponding block, thereby obtaining the lighting information for all blocks that need updating.

To avoid an excessive number of blocks needing updates in a single frame, we adopt a frame-splitting mechanism that limits the number of blocks that can be updated per frame. For each frame, only the first 64 blocks requiring updates are processed, while the remaining blocks are queued for the next frame. This approach distributes the computational load evenly across frames to reduce per-frame overhead. Additionally, we partition the target blocks using a multithreading approach, allowing for parallel processing of all blocks that require updates.

Once we have the blocks required for the current frame, we need to update the corresponding volumes with these blocks. We use a scroll manner to index each block within the volume, as illustrated in Figure 4 (for simplicity, we only use one level of volume). This allows us to avoid the redundant block updating.

At the same time, the discarded blocks may also be reused. After obtaining the indirect lighting information for each block, we store the computed results in an LRU cache. As the viewer's path in the scene typically exhibits locality, caching blocks in key areas can effectively reduce computational pressure. Finally, we organize the volume indexed by the scroll coordinates into a fully usable ordinary index for pixels through a single parallel computation in the GPU.

4 Implementation

We utilize the PyTorch framework for the majority of the optimization computations and write custom CUDA kernels, inspired by the gsplat [32] and GaussianImage [35] project, to facilitate rapid inference of coefficients for a given probe and the corresponding gradient backpropagation. To evaluate our approach, we extract precomputed probe data from the open-source Unreal Engine [7], compress it externally and assess the compression ratio of our algorithm. After compression, we re-import the data back into the engine and integrate CLV algorithm to achieve runtime decompression. We directly utilize Unreal Engine's implementation of the octree and LRU Cache. The LRU Cache identifies each block by its XYZ index and LOD level, while the octree stores the positions of all probes to facilitate rapid identification of probes within a block during decompression. For real-time decompression, we implement parallel computation of Gaussian functions and network inference via compute shaders. This pipeline resembles our training process. Notably, we pack weight matrix as 4x4 matrix to optimize register utilization. All our experiments are conducted on a platform equipped with Intel i7-13700 CPU, 64GB of system memory, and NVIDIA RTX 4060 GPU.

5 Experiments

Our compression scheme allows for the adjustment of various including the number of Gaussians, the quantization bits, the latent feature dim, and the size of the decoding network. In all of our experiments, the decoding network consists of two hidden layers, each with a width of 64. The activation function is set to Sigmoid. In our approach, the compression quality is primarily adjusted by varying the proportion of Gaussian functions in the probe count. For most experiments, this ratio is set to 0.05, with the latent feature dimension fixed at 9 and the quantization bit depth configured as 10. Our experiments indicate that this is generally sufficient to reconstruct the ideal light field. Across all test scenes, we use a learning rate of 0.01. For the Gaussian model, we employ the Adan [31] optimizer, while the decoder uses the Adam [13] optimizer. We train for 10,000 iterations followed by 2,000 iterations of quantization training.

With respect to CLV, we use 4 volumes with block size 2m, 4m, 8m and 32m. Each volume has 16^3 blocks. In all of our experiments, CLV is used to exploit the light field probe data and render the test image, no matter the probe is compressed or not.

We constructed several large-scale test environments for compression experiments. This dataset includes common landscapes such as cities, villages, islands, and factories, covering different times of day from daylight to nighttime. In each scene, a fixed-size area was selected, and uniformly distributed light field probes were generated on the object surfaces in this area. All test scenes contain approximately 15,000 probes each.

5.1 Compared Methods

We conducted qualitative and quantitative comparisons with a series of traditional PCA-based methods, including global PCA (GPCA), blockwise PCA (BPCA) [21], windowed blockwise PCA (WBPCA) [21] and clustered PCA (CPCA) [25]. Global PCA directly extracts the principal components from the high-dimensional data of all probes. BPCA, on the other hand, employs fixed-size blocks to extract the PCA basis. WBPCA builds upon BPCA by expanding the range used for principal component calculation to include surrounding blocks, thereby mitigating the discontinuities that can arise at block boundaries. CPCA first clusters the probes based on their positions, then performs independent calculations within each cluster. Our method demonstrates significant improvements in both the compression ratio and decompression quality compared to previous approaches.

5.2 Quantitative Results

In our experiments, we randomly select multiple viewpoints and use probe data compressed by various methods, integrating them with our CLV algorithm for rendering. The rendered images are compared with those rendered using the original probe data to calculate PSNR and SSIM. Table 1 illustrates the significant advantages of our method at different compression rates compared to other techniques. We attribute these improvements to the application of

Gaussian functions to adaptively represent the latent field of the probe data and the use of neural networks as universal approximators [9]. We qualitatively demonstrate the compression accuracy of our method in Figure 5. Compared to CPCA [21], our method exhibits no significant visible errors across various scenes.

Table 1. Average MSE, ABS, PSNR, SSIM values across the test scenes. The comparison is divided into two groups by their compression ratio. "bpp" indicates bits per probe. We employ 5% Gaussian functions under low compression rates and 20% under high compression rates, while keeping all other parameters unchanged.

	Low (2~3%)					**High (6~7%)**				
	GPCA	BPCA	WBPCA	CPCA	Ours	PCA	BPCA	WBPCA	CPCA	Ours
MSE (↓)	0.1228	0.0920	0.1031	0.0621	**0.0133**	0.0551	0.0394	0.0429	0.0291	**0.0045**
ABS (↓)	0.1552	0.1367	0.1467	0.1097	**0.0649**	0.1072	0.0928	0.0978	0.0748	**0.0388**
PSNR (↑)	35.17	37.00	35.92	39.11	**42.91**	39.22	40.85	39.92	42.02	**44.82**
SSIM (↑)	0.9845	0.9872	0.9853	0.9901	**0.9923**	0.9894	0.9910	0.9902	0.9923	**0.9929**
bpp	20.11	22.36	23.26	29.75	21.01	58.21	55.34	61.50	62.63	52.97
Ratio(%)	2.33	2.59	2.69	3.44	2.43	6.73	6.40	7.11	7.25	6.13

5.3 Performance

In this section, we discuss the compression performance and real-time decompression performance of GPC.

5.3.1 Compression. The compression time includes both the initial training phase and subsequent quantization fine-tuning. We implemented Gaussian function inference and backpropagation using custom CUDA kernels. As shown in the Table 2, the training time increases with the ratio of Gaussian functions relative to the probe count (i.e., the number of Gaussian functions). Additionally, the expanding coverage area of Gaussian functions during training leads to progressively slower inference speeds. Optimizing redundancies in the forward pass (Figure 3), such as adopting more precise methods for calculating Gaussian coverage areas, could potentially enhance compression speed. For comparision, traditional PCA methods (GPCA, BPCA) complete compression within one second, CPCA requires additional clustering steps, resulting in compression time of several seconds. This results in our method being approximately one order of magnitude slower than PCA-based approaches.

Table 2. The average compression time across all test scenarios, increasing with the ratio of Gaussian functions.

ratio	GPC 0.02	GPC 0.05	GPC 0.10	GPC 0.15	GPC 0.20
time	48.68s	64.60s	87.41s	108.27s	125.56s

5.3.2 Decompression. The primary computational overhead of decompression lies in feature calculation for each probe and subsequent MLP inference. To measure the maximum per-frame overhead, we deliberately disabled the LRU cache, ensuring all probe data required for block updates per frame were decompressed in real-time using our method. The decompression speed primarily depends on the number of Gaussian functions in the scene. As shown in Table 3, the inference time per frame remains constrained within a low range. Moreover, since the network's computational load only relates to the number of probes decompressed per frame, it maintains stable performance across all compression ratios. We store both Gaussian parameters and network weights in half-precision floating-point format, and calculate their total memory consumption at various compression ratios.

Table 3. Decompression performance in the OldTown scene. The decompression performance of our model is similar across all test scenes for a given Gaussian functions ratio.

Process	GPC 0.02	GPC 0.05	GPC 0.10	GPC 0.15	GPC 0.20
Gaussion Infer.	0.26 ms	0.44 ms	0.82 ms	1.16 ms	1.33 ms
MLP Infer.	0.04 ms	0.04 ms	0.04 ms	0.04 ms	0.04 ms
Sum	0.30 ms	0.48 ms	0.86 ms	1.20 ms	1.37 ms
Memory	27.95KB	50.1KB	87.03KB	123.95KB	160.87KB

6 Discussion

In this section, we first discuss the limitations of the proposed compression method and analyze possible future directions.

6.1 Limitations

Dynamic scenes. Our method inherits the inherent limitations of precomputed light probe approaches. While light probes primarily provide global illumination effects for dynamic objects within scenes, as precomputed data they cannot provide real-time respond to illumination changes caused by dynamic objects, thus not supporting dynamic scene representation.

Training Speed. Our method requires compression times on the order of minutes and is significantly slower than PCA-based approaches that complete within seconds. With the majority of the training time being spent on inference in PyTorch, integrating all processes into CUDA, similar to NTC [29], could lead to more efficient training. In scenes lacking luminance and chromatic variations, PCA is still a more preferable alternative both for compression and decompression.

6.2 Future Work

Time of Day. It is often necessary to restore the lighting variations of a scene throughout the day, referred to as time of day (TOD). An interesting direction for future research is whether we can leverage neural networks to extract temporal redundancy from TOD probe data, thereby achieving a broader range of compression.

Image Domain Optimization. While the accuracy of decompressing probes can determine the final image quality, we also aim to explore new optimization pathways, such as directly optimizing the rendering results obtained from compressed probes. This approach could minimize the impact of the compression process on the visual output more directly and we leave this as a topic for future work.

Further Applications. Beyond its immediate application to radiance compression, our proposed mixture of neural Gaussians representation exhibits broader potential for general spatial signal encoding tasks similar to hash grids from InstantNGP [19] and triplanes from EG3D [4], which became foundational building blocks for neural rendering and 3D content generation. Our hybrid Gaussian-MLP framework could extend to real-time neural rendering where compactness and decoding speed are critical (e.g., replacing hash grids)

or point cloud compression, where existing methods struggle with non-uniform geometry and high-dimensional attributes.

7 Conclusion

We propose a novel method specifically designed for compressing light probe data. We employ neural networks to extract the latent features of the probes and adaptively encode the spatial correlations of the probes using Gaussian functions, significantly surpassing the conventional PCA-based approaches. By leveraging custom CUDA kernels for Gaussian function inference and updates, we achieve rapid probe data compression and fast real-time decompression. This substantially reduces both disk and memory usage for light probe data. To address the storage and memory demands of conventional 3D light field textures, we propose cascaded lighting volumes which eliminates the need for dense probe placement and further reduce light probe memory consumption.

We hope our work will assist more performance-constrained devices in rendering high-quality global illumination, and inspire further exploration of neural representations for illumination.

References

[1] Martin Balint, Krzysztof Wolski, Karol Myszkowski, Hans-Peter Seidel, and Rafał Mantiuk. 2023. Neural Partitioning Pyramids for Denoising Monte Carlo Renderings. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 60, 11 pages. https://doi.org/10.1145/3588432.3591562

[2] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517. https://doi.org/10.1145/361002.361007

[3] Rasmus Bro and Age K Smilde. 2014. Principal component analysis. *Analytical methods* 6, 9 (2014), 2812–2831.

[4] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. 2022. Efficient Geometry-aware 3D Generative Adversarial Networks. In *CVPR*.

[5] Hajin Choi, Seokpyo Hong, Inwoo Ha, Nahyup Kang, and Bochang Moon. 2024. Online Neural Denoising with Cross-Regression for Interactive Rendering. *ACM Trans. Graph.* 43, 6, Article 221 (Nov. 2024), 12 pages. https://doi.org/10.1145/3687938

[6] Robert Cupisz. 2012. Light probe interpolation using tetraheral tessellations. https://www.gdcvault.com/play/1015312/Light-Probe-Interpolation-Using-Tetrahedral.

[7] Epic Games. [n. d.]. *Unreal Engine*. https://www.unrealengine.com

[8] G. Greger, P. Shirley, P.M. Hubbard, and D.P. Greenberg. 1998. The irradiance volume. *IEEE Computer Graphics and Applications* 18, 2 (1998), 32–43. https://doi.org/10.1109/38.656788

[9] K. Hornik, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Netw.* 2, 5 (July 1989), 359–366.

[10] Henrik Wann Jensen. 1996. Global illumination using photon maps. In *Rendering Techniques' 96: Proceedings of the Eurographics Workshop in Porto, Portugal, June 17–19, 1996 7*. Springer, 21–30.

[11] Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Washington, D.C.) *(I3D '10)*. Association for Computing Machinery, New York, NY, USA, 99–107. https://doi.org/10.1145/1730804.1730821

[12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)* 42, 4 (July 2023). http://www-sop.inria.fr/reves/Basilic/2023/KKLD23

[13] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] https://arxiv.org/abs/1412.6980

[14] Jaroslav Křivánek, Pascal Gautron, Greg Ward, Okan Arikan, and Henrik Wann Jensen. 2007. Practical global illumination with irradiance caching. In *ACM SIGGRAPH 2007 courses*. 1–es.

[15] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[16] Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. 2019. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques* 8, 2 (2019).

[17] Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. 2017. Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (San Francisco, California) *(I3D '17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. https://doi.org/10.1145/3023368.3023378

[18] 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.

[19] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

[20] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.* 40, 4, Article 36 (July 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

[21] K. Nishino, S.K. Nayar, and T. Jebara. 2005. Clustered blockwise PCA for representing visual data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 10 (2005), 1675–1679. https://doi.org/10.1109/TPAMI.2005.193

[22] Yaobin Ouyang, Shiqiu Liu, Markus Kettunen, Matt Pharr, and Jacopo Pantaleoni. 2021. ReSTIR GI: Path resampling for real-time path tracing. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 17–29.

[23] Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global illumination with radiance regression functions. *ACM Trans. Graph.* 32, 4 (2013), 130–1.

[24] Ari Silvennoinen and Peter-Pike Sloan. 2021. Moving basis decomposition for precomputed light transport. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 127–137.

[25] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. 2003. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 382–391.

[26] Peter-Pike Sloan, Jan Kautz, and John Snyder. 2023. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 339–348.

[27] Natalya Tatarchuk, Jonathan Dupuy, Thomas Deliot, Daniel Wright, Krzysztof Narkowicz, Patrick Kelly, Aleksander Netzel, and Tiago Costa. 2022. Advances in real-time rendering in games: part I. In *ACM SIGGRAPH 2022 Courses* (Vancouver, British Columbia, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 18, 1 pages. https://doi.org/10.1145/3532720.3546895

[28] Unity Technologies. [n. d.]. *Unity Real-Time Development Platform.* https://www.unity.com

[29] Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Möller, Pontus Ebelin, and Aaron Lefohn. 2023. Random-access neural compression of material textures. *arXiv preprint arXiv:2305.17105* (2023).

[30] Yue Wang, Soufiane Khiat, Paul G Kry, and Derek Nowrouzezahrai. 2019. Fast non-uniform radiance probe placement and tracing. *I3D* 19 (2019), 21–23.

[31] Xingyu Xie, Pan Zhou, Huan Li, Zhouchen Lin, and Shuicheng Yan. 2024. Adan: Adaptive Nesterov Momentum Algorithm for Faster Optimizing Deep Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[32] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. 2024. gsplat: An Open-Source Library for Gaussian Splatting. *arXiv preprint arXiv:2409.06765* (2024). arXiv:2409.06765 [cs.CV] https://arxiv.org/abs/2409.06765

[33] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. 2019. Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets. arXiv:1903.05662 [cs.LG] https://arxiv.org/abs/1903.05662

[34] Song Zhang*, Daqi Lin*, Markus Kettunen, Cem Yuksel, and Chris Wyman. 2024. Area ReSTIR: Resampling for Real-Time Defocus and Antialiasing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2024)* 43, 4, Article 98 (07 2024), 13 pages. https://doi.org/10.1145/3658210 (*Joint First Authors).

[35] Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. 2024. GaussianImage: 1000 FPS Image Representation and Compression by 2D Gaussian Splatting. *arXiv preprint arXiv:2403.08551* (2024).

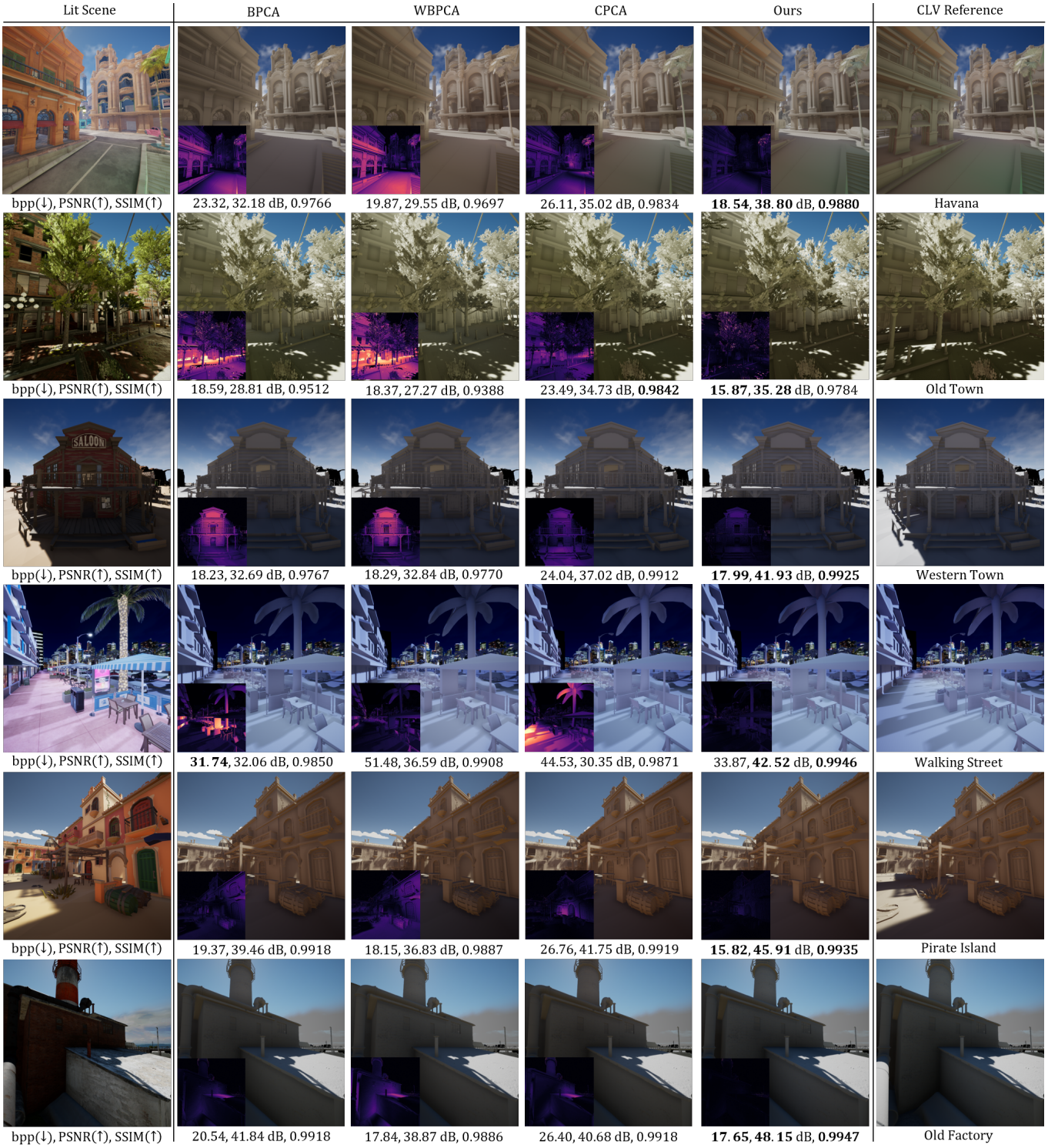

Fig. 5. Qualitative comparison results from test scenes. We compare our method with BPCA [21], WBPCA [21], and CPCA [25] under low compression ratio. Lit Scene denotes the fully rendered reference scene, while other results are rendered without surface textures to better visualize incident lighting.

.