EUROGRAPHICS 2026 / B. Masia and J. Thies
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 45 (2026), Number 2

Lightmap Compression with Color-Coherent UV Clustering and Cascade Texture Optimization

Dehan Chen[1], Hongyu Huang[1], Yuzhe Luo[1], Hao Xu[1], Yuqing Zhang[1], Sipeng Yang[1], Xifeng Gao[3], Heng Cai[2], Chao Li[2†] and Xiaogang Jin[1†]

[1]State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, China,
[2]Tencent, [3]LIGHTSPEED

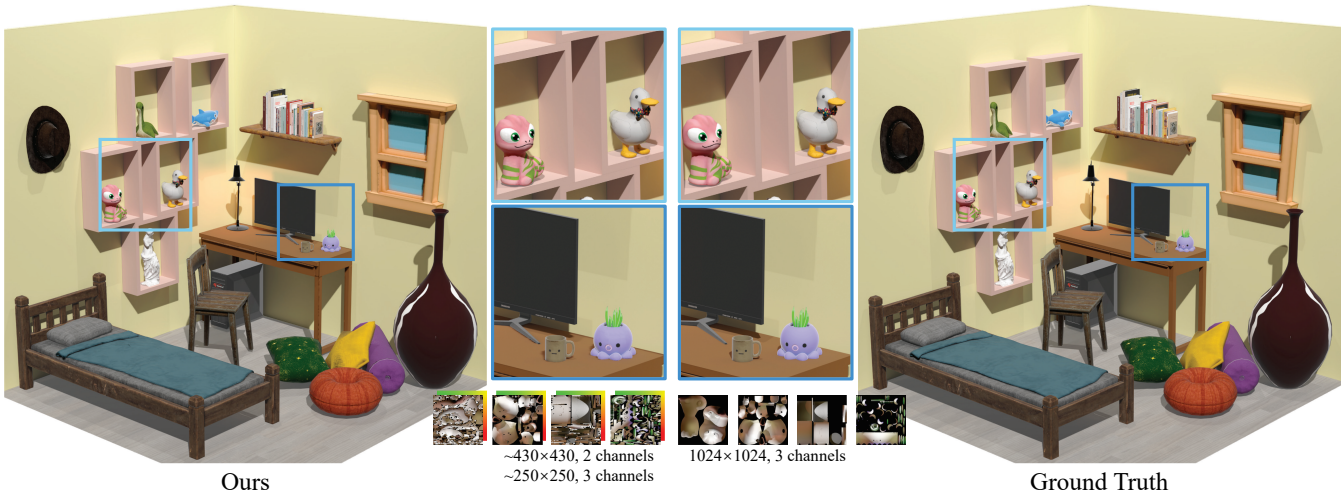


Figure 1: *Comparison of a bedroom scene rendered with our compressed lightmaps (left) and with ground-truth lightmaps (right) under bright illumination. Zoomed-in regions illustrate the preservation of fine structural details, and a set of representative lightmap textures is shown to contrast the pre- and post-compression results. Overall, our method achieves an* ***83% reduction in storage cost****, with negligible perceptual differences as reflected by quantitative metrics* ***(PSNR 51.79 dB, 1–SSIM 0.0027)****.*

Abstract

To address the storage overhead of lightmaps and the limitations of existing compression techniques, we propose a novel UV-space compression framework based on per-triangle processing. By mapping triangles to a standardized domain, we cluster and repack color-coherent regions into a compact atlas, generating a cascade texture refined via differentiable rendering. Experimental results show an average storage reduction of 83% with approximately 10 dB higher PSNR than existing methods. Our approach is the first dedicated lightmap compression framework compatible with standard block-based formats, offering an effective solution for memory-efficient 3D asset delivery.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics;***

1. Introduction

Lightmap is a widely used technique in real-time rendering for efficiently approximating global illumination. It is typically generated through radiosity [CWH93] computation, which captures the surface lighting of a scene. By storing this information in texture maps and sampling in shading process, lightmapping greatly reduces runtime shading costs and ensures stable performance, even in large-scale or resource-constrained applications. Nevertheless, high-resolution lightmaps introduce significant storage, memory,

† Corresponding authors

and bandwidth overhead—often becoming a dominant bottleneck in asset delivery and runtime memory usage, especially for complex scenes with dense occlusion and multiple light bounces.

Compressing lightmaps presents unique challenges compared to conventional image compression. First, lightmap compression must support efficient random access, allowing lighting information for any pixel or region to be quickly retrieved without decompressing the entire texture, which traditional formats such as JPEG [Wal92] cannot provide. Second, lightmaps are characterized by large-scale low-frequency illumination patterns and strong shading coherence across spatially separated but geometrically adjacent triangles—features arising from global illumination effects like soft shadows and interreflection. However, existing industry-standard block-based texture compression algorithms, such as DXT series [Mic20], ETC2 [SP07], and ASTC [NLP*12], operate locally within small fixed-size blocks (e.g., 4×4 pixels), failing to exploit this long-range coherence. Complementary approaches [LJP*23, KG25] attempt to reuse UV coordinates for uniformly shaded regions, combining UV-space optimization with compression. Yet, due to the intrinsic smoothness of lightmap shading even within individual triangles, such methods achieve only modest gains.

To address these challenges, we propose a lightmap compression framework centered on color-coherent UV clustering and cascade texture optimization. By jointly leveraging shading similarity and multi-resolution encoding, our method achieves significant storage reduction while maintaining high visual fidelity and compatibility with real-time rendering pipelines.

Specifically, to handle the smooth color gradients within triangles, which make similarity estimation more challenging than in prior methods that treat triangles as single solid colors [LJP*23], we first normalize each triangle to a canonical isosceles right triangle and store it in a feature texture. This standardized representation allows pixel-wise comparison of shading patterns, enabling accurate similarity measurement. We compute pairwise similarities to construct a graph, where nodes represent triangles and edges encode similarity. A greedy graph partitioning algorithm then clusters highly similar triangles, which are repacked into shared UV regions. This UV repacking yields a compact, lower-resolution lightmap. We further replace it with a cascade texture representation for higher compression efficiency. Finally, differentiable rendering refines this hierarchy to eliminate seams and preserve visual fidelity.

Our framework is fully compatible with existing block-based compression algorithms and can be seamlessly integrated into real-time rendering systems. In experiments, it achieves an average lightmap storage reduction of 83%, with a render-space PSNR of 51.79 dB and 1–SSIM of 0.0027, demonstrating both high compression efficiency and strong visual fidelity. Our contributions are summarized as follows:

- We present the first dedicated framework for lightmap compression, explicitly designed to exploit UV-space coherence while remaining compatible with block-based texture formats and real-time rendering pipelines.
- We develop a feature texture construction scheme that geometrically standardizes irregular UV triangles into canonical isosceles right triangles, enabling accurate pixel-wise similarity assessment.
- We introduce the first integration of cascade texture into a differentiable rendering pipeline, achieving substantial resolution reduction with view-consistent seam removal and high-fidelity preservation.

2. Related Work

In this section, we review prior work in two key areas: image compression and texture compression. Our method focuses on lightmap texture compression, addressing the limitations of existing approaches and targeting the specific challenges posed by lightmaps.

2.1. Image Compression

Image compression reduces storage requirements and transmission bandwidth of digital images and is widely used in modern applications. Methods can be divided into two main categories: traditional algorithms and neural network-based approaches, each addressing different trade-offs among compression efficiency, image quality, and computational complexity.

Traditional image compression techniques reduce storage and transmission costs using fixed algorithms based on mathematical transformations or heuristics, and have been extensively optimized over decades. For example, JPEG [Wal92] employs the Discrete Cosine Transform (DCT) for lossy compression, exploiting spatial frequency characteristics to remove perceptually less important information while maintaining visual fidelity. JPEG XL [AvAB*19] extends JPEG with lossy and lossless modes, smaller file sizes, faster decoding, and improved high-resolution performance. AVIF [CMH*18], based on the AV1 codec, further enhances compression efficiency and supports modern features including High Dynamic Range (HDR), alpha transparency, and higher bit depths. These methods remain essential when computational resources are limited or compatibility with legacy systems is required.

Recently, neural network-based image compression methods have emerged to improve both compression efficiency and perceptual quality. These works apply deep learning models—such as GANs [ATM*19], VAEs [BMS*18], and diffusion based models [GPW*23]—to the task of image compression, learning compact, task-specific representations that enable higher compression ratios while minimizing perceptual loss. By leveraging these learned representations, neural methods achieve higher compression ratios while minimizing perceptible loss. Subsequent works have further enhanced neural compression architectures. For example, Agustsson et al. [AMTM23] propose a flexible decoder for dynamic control over the distortion–realism trade-off, Muckley et al. [MENU*23] improve statistical fidelity via a non-binary discriminator on quantized VQ-VAE representations, and the ELIC model [HYP*22] introduces space–channel adaptive coding to boost compression performance. Additionally, Emilien [DGA*21] and Strümpler [SPY*22] explore storing MLP weights for efficient decoding.

Despite their strengths, both traditional and neural image compression methods generally lack support for random access, which

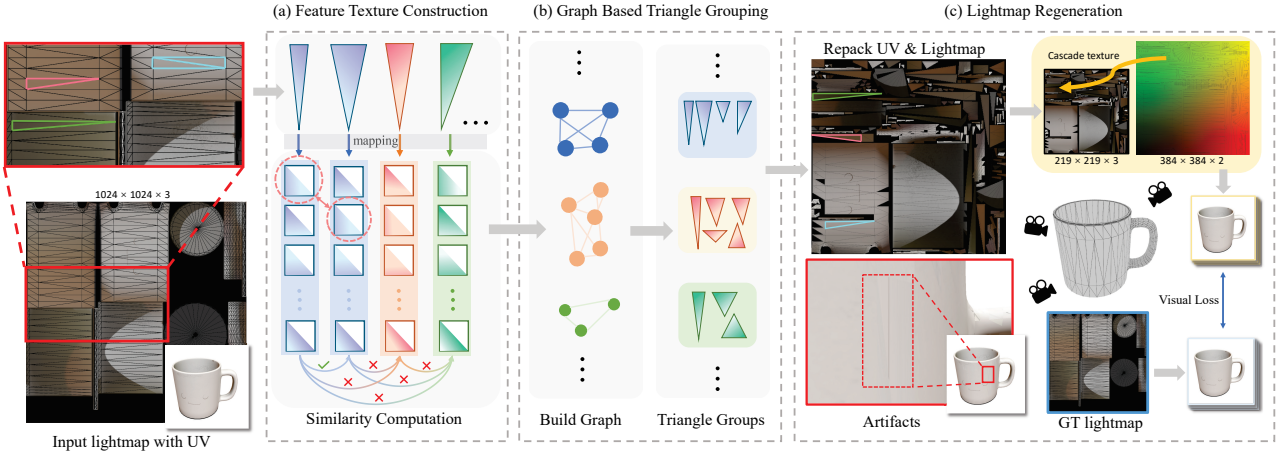


Figure 2: *Overview of our pipeline. (a) Each UV triangle is mapped to a unit square feature texture via barycentric parameterization, producing normalized patches for reliable color comparison. (b) Triangles with similar shading patterns are grouped via greedy clustering, and representative ones are used to pack shared UV regions. (c) A compact UV layout is first generated through rigid repacking, enabling a reduced-resolution lightmap; this is followed by joint optimization of the UV LUT and color texture using differentiable rendering to ensure seamless and visually faithful reconstruction. This dual-texture approach achieves higher compression ratio while preserving visual quality.*

is essential for lightmap texture compression. By contrast, our method achieves high compression efficiency while enabling efficient random access to arbitrary texture regions, making it well suited for real-time rendering pipelines.

2.2. Texture Compression

Texture compression is distinguished from general image compression by its requirement to support random access to specific regions, which is essential for real-time rendering applications. Early block-based methods established the basis for modern texture compression techniques. Block Truncation Coding (BTC), proposed by Delp et al. [DM79], introduced the division of images into fixed 4×4 pixel blocks with quantization applied to each block, significantly reducing storage requirements. Building on this idea, S3TC [Wik25] extended block-based compression to multi-channel images by encoding each 4×4 block with two 16-bit color values and 2-bit indices per pixel, which laid the groundwork for the widely adopted DXT/BC series [Mic20], with BC1 being one of the most popular formats.

Subsequent methods refined block-based compression techniques to accommodate platform-specific constraints. ETC (Ericsson Texture Compression) targets mobile and embedded devices, with ETC1 [SAM05] subdividing each 4×4 block into smaller regions (e.g., 2×4 or 4×2) to allow different color endpoints and interpolation modes, achieving a balance between compression efficiency and visual quality. ETC2 [SP07] extends ETC1 by supporting RGBA textures and refining encoding modes for higher color fidelity, particularly in transparent regions. ASTC (Adaptive Scalable Texture Compression) [NLP*12], introduced by ARM, generalizes block-based compression with variable block sizes ranging from 4×4 to 12×12 pixels, allowing flexible trade-offs between compression ratio and visual fidelity. In contrast, PVRTC (PowerVR Texture Compression) [Fen03], developed by Fenney, combines fixed-size block partitioning with frequency-based modulation of high- and low-frequency components, providing efficient compression while maintaining visual fidelity, even at low bitrates. While these classical methods perform well in real-time rendering scenarios, they mainly focus on hand-crafted block encodings and have limited capacity to exploit redundancies specific to the data, particularly in images with large-scale low-frequency features such as lightmaps. This limitation has motivated recent studies on neural network-based texture compression, aiming to learn compact, high-fidelity representations that preserve random access efficiency.

Neural texture compression has focused on compact representations with efficient random access. Vaidyanathan et al. [VSW*23] introduced Neural Texture Compression (NTC), the first method to use a hierarchical feature grid with an MLP for real-time decompression, though it is limited to fixed resolutions among texture sets. Farhadzadeh et al. [FHL*24] extended it to multi-resolution textures, while Datta et al. [DMD*23] generalized the grid structure to other graphics primitives. In contrast, our method requires no neural network inference at runtime.

A complementary direction, more aligned with our approach, shifts focus from neural representation to geometric redundancy in UV space. Methods such as Luo et al. [LJP*23] merge single-color triangles, and Knodt et al. [KG25] share features across mirror-symmetric UV charts, both reducing texture size by exploiting structural regularities in UV layouts. These approaches align compression with rasterization geometry but are less effective for lightmaps, which typically lack large single-color regions and exhibit irregular, non-symmetric UV layouts, thus limiting the utility of such redundancy-based strategies.

3. Method

Given a 3D mesh $\mathcal{M}$ containing vertex coordinates $\mathbf{V}$, UV coordinates $\mathbf{U}$, and pre-baked HDR lightmap $\mathcal{T}_{\text{orig}}$, we propose a novel

three-stage compression framework to generate compressed cascade textures with new color texture $\mathcal{T}_p$ and UV LUT texture $\mathcal{T}_c$. As illustrated in Figure 2, our pipeline first constructs feature textures for all UV triangles through barycentric mapping, enabling robust color similarity assessment (Section 3.1). Next, we partition triangles into similarity groups via greedy graph clustering, consolidating color-coherent regions into shared UV spaces using representative triangles (Section 3.2). Finally, we repack and generate optimized cascade textures through differentiable rendering, where UV LUT texture $\mathcal{T}_c$ guides sampling from color texture $\mathcal{T}_p$ via triangle-wave mapping $\mathcal{F}(x)$(Section 3.3). This dual-texture system achieves aggressive compression while eliminating seam artifacts through view-consistent optimization.

3.1. Feature Texture Construction

Lightmap baking frequently yields perceptually similar triangles due to gradual illumination gradients across surfaces. However, conventional non-overlapping UV parametrization isolates each triangle in distinct texture regions, creating significant redundancy when chromatically coherent triangles are spatially separated. To exploit this geometric redundancy, we introduce a normalization pipeline that projects irregular UV triangles onto isosceles right triangles within square domains. This transformation enables rigorous color similarity assessment while accommodating arbitrary geometric configurations.

The core insight is that any UV triangle Δ_i with vertices $\{\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2\}$ (where each $\mathbf{p}_k = (u_k, v_k)$ denotes a 2D UV coordinate) can be standardized within a square domain. This geometric normalization requires calculating an optimal texture resolution R_s that preserves color fidelity:

$$R_s = \lceil R \cdot \max\left(\|\mathbf{p}_0 - \mathbf{p}_1\|, \|\mathbf{p}_1 - \mathbf{p}_2\|, \|\mathbf{p}_2 - \mathbf{p}_0\|\right) \rceil, \quad (1)$$

where R denotes the resolution of $\mathcal{T}_{orig}$. This formulation ensures complete coverage of the triangle's UV extent while minimizing padding overhead.

To populate the $R_s \times R_s$ texture $\mathcal{T}_s$, we map each pixel in the square domain to a corresponding position inside the original UV triangle using barycentric interpolation. This is done by aligning the triangle's vertices to fixed corners of the square, converting pixel coordinates into barycentric weights, and sampling the original lightmap via bilinear interpolation. Only pixels within the lower-triangular region (corresponding to valid barycentric coordinates) are assigned colors; the rest are left empty. Although the texture remains a full square in memory, this layout provides a shape-agnostic, standardized representation that eliminates geometric redundancy and enables consistent similarity comparison across arbitrary triangles.

Crucially, vertex ordering ambiguity can cause triangles with identical color distributions to appear dissimilar under inconsistent correspondences—e.g., $\triangle ABC$ and $\triangle DEF$ may seem different under $A \to D, B \to E, C \to F$ but similar under $A \to F, B \to E, C \to D$. To ensure robust, topology-agnostic similarity evaluation, we generate six feature textures per triangle, covering all vertex permutations, as shown in Figure 3. This yields a permutation-invariant representation $\{\mathcal{T}_t^{(i)}\}_{t=0}^{5}$ for each UV triangle $\triangle i$, forming the foundation for our graph-based grouping in Section 3.2.

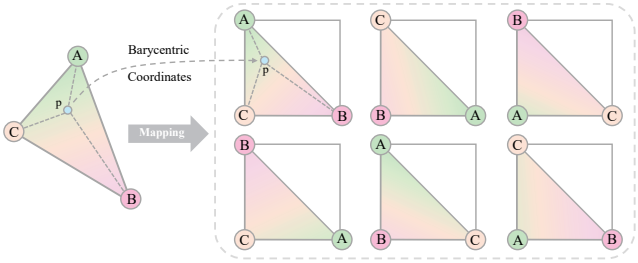


Figure 3: *Feature texture construction. Each triangle is mapped to the lower-triangular domain of a square feature texture, producing six possible configurations corresponding to vertex permutations. Pixel colors within the domain are obtained via bilinear interpolation of the original triangle using barycentric coordinates.*

3.2. Similarity Graph Based Triangle Grouping

Building upon the per-triangle square textures constructed in Section 3.1, we formalize a graph-based grouping algorithm to cluster triangles with similar chromatic properties. This phase consists of two sequential components: (1) constructing a similarity graph where edges encode color affinity relationships, and (2) partitioning the graph to identify cohesive groups of triangles that can share unified UV regions. The grouping strategy reduces redundancy in lightmap representation while preserving visual fidelity, enabling efficient UV repacking in Section 3.3.

3.2.1. Similarity Graph Construction

The similarity assessment begins with a computationally efficient pruning strategy based on dominant color analysis. For each triangle, we construct a color histogram of its covered pixels using 32 bins per RGB channel and identify the center of the most frequent bin as its dominant color. When comparing two triangles $\triangle_i$ and $\triangle_j$, if the Euclidean distance between their dominant colors exceeds threshold λ_h, we immediately classify them as dissimilar ($S_{\text{final}} = -1$) without further computation. This preliminary filtering significantly reduces unnecessary processing.

For triangle pairs passing the dominant color test, we perform rigorous color comparison. Let $T_i^{(k)}$ and $T_j^{(l)}$ denote the square textures of triangles $\triangle_i$ and $\triangle_j$ respectively, where $k, l \in \{0, 1, \ldots, 5\}$ enumerate the six vertex permutations. The L_1 distance between textures is computed as:

$$\text{Dist}(T_i^{(k)}, T_j^{(l)}) = \frac{1}{N} \sum_{p=1}^{N} \left| T_i^{(k)}[p] - T_j^{(l)}[p] \right|, \quad (2)$$

where N is the pixel count. To ensure resolution invariance, we upsample smaller textures via bilinear interpolation to match larger counterparts. The minimal distance across vertex correspondences determines similarity:

$$d_{\min}(i, j) = \min_{l \in \{0,1,\ldots,5\}} \text{Dist}(T_i^{(0)}, T_j^{(l)}). \quad (3)$$

The final similarity status is then thresholded:

$$S_{\text{final}} = \begin{cases} \arg\min_{l} \text{Dist}(T_i^{(0)}, T_j^{(l)}), & d_{\min}(i,j) < \lambda \\ -1, & d_{\min}(i,j) \geq \lambda \end{cases} \tag{4}$$

where λ is a perceptually calibrated similarity threshold.

This pairwise assessment generates an adjacency matrix $\mathbf{A} \in \{-1, 0, \ldots, 5\}^{M \times M}$, where M denotes the total number of triangles. It forms an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where:

$$\begin{aligned} \mathcal{V} &= \{\triangle_i \mid i = 1, \ldots, M\}, \\ \mathcal{E} &= \{e_{ij} \mid S_{\text{final}}(i,j) \neq -1\}. \end{aligned} \tag{5}$$

This graph structure efficiently encodes the chromatic relationships among the triangles, as illustrated in Figure 2 (b).

3.2.2. Graph Partitioning via Greedy Reduction

To consolidate color-coherent regions, we partition $\mathcal{G}$ using a greedy algorithm that prioritizes high-connectivity nodes. The process initiates by selecting the node v^* with maximum degree:

$$v^* = \arg\max_{v \in \mathcal{V}} \deg(v). \tag{6}$$

We form group $\mathcal{G}^* = \{v^*\} \cup \mathcal{N}(v^*)$ containing the node and its first-order neighbors. After removing $\mathcal{G}^*$ from the graph, the process iterates until all edges are exhausted.

Within each identified group $\mathcal{G}_k$, the representative triangle is chosen as the one with the highest degree in that group, thereby maximizing color consistency within the shared UV region, while singleton groups retain their original triangle as representative. This iterative reduction process, illustrated in Figure 4, yields compact sets $\{\mathcal{G}_k\}$ with representatives $\{\triangle_{\text{rep}}^{(k)}\}$, effectively consolidating UV space while preserving perceptual quality.

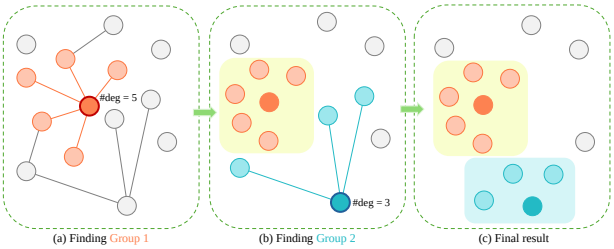


Figure 4: *Triangle grouping process. (a) The node with maximum degree 5 is selected as representative, together with its first-order neighbors forming Group 1. (b) After removing Group 1 nodes, the node with maximum degree 3 becomes representative, forming Group 2 with its neighbors. (c) With all edges eliminated, each isolated node serves as its own representative, yielding 6 groups in total.*

3.3. Lightmap Regeneration

3.3.1. UV Repacking and Lightmap Generation

Building upon the triangle grouping framework established in Section 3.2, we now address UV repacking and lightmap regeneration. Each group, represented by a triangle with maximal similar edges, serves as the foundation for UV realignment. Non-representative triangles within each group undergo rigid transformations to align their UV coordinates with the representative triangle, guided by the similarity status S_{final} computed between each pair of non-representative triangle and the representative, thereby preserving structural integrity while minimizing deviation from the original UV layout.With these aligned UV coordinates, the next step is to organize them efficiently within a UV atlas to generate the repacked lightmap.

To accomplish this, we adapt the standard UV packing pipeline from XAtlas [You17], which typically involves cutting, parameterization, and packing stages. However, to prevent distortion of the original UV mapping, we bypass the parameterization phase and focus exclusively on rigid transformations during packing. This approach maintains geometric fidelity while ensuring consistent texture sampling across grouped triangles.

The repacked UV atlas enables generation of a new lightmap at reduced resolution. We introduce a hyperparameter ϵ as a scaling factor, defining the regenerated lightmap resolution as $W_r \times H_r = \epsilon W \times \epsilon H$, where W and H denote the original lightmap dimensions. This downscaling preserves the aspect ratio of each representative triangle, ensuring that the UV geometry is uniformly scaled without stretching or squeezing, while retaining at least ϵ-times the original sampling density.

To populate the repacked lightmap, each pixel (x, y) is processed as follows:

1. Compute barycentric coordinates within its host triangle
2. Map these coordinates to the original lightmap space using the established correspondences
3. Sample the color via bilinear interpolation

This process preserves the original lighting information as faithfully as possible while accommodating the repacked UV layout. The regenerated lightmap maintains visual fidelity despite resolution reduction, leveraging the chromatic coherence established during grouping to minimize artifacts.

3.3.2. Cascade Texture Generation

The UV repacking process described in Section 3.3.1 achieves initial compression but presents opportunities for further optimization. To enhance compression ratios while maintaining visual fidelity, we adopt the hierarchical array structure proposed by the Differentiable Indirection method [DMD*23], and build a similar cascade texture representation. This representation consists of:

- *UV LUT texture* (T_c): Encodes remapped UV coordinates with 2 channels at resolution $W_c \times H_c = \epsilon_c W_r \times \epsilon_c H_r$.
- *Color texture* (T_p): Stores color values with 3 channels at reduced resolution $W_p \times H_p = \epsilon_p W_r \times \epsilon_p H_r$.

where ϵ_c and ϵ_p are compression factors. This design captures low-frequency color variations in the low-resolution color texture, while the high-resolution UV LUT modulates these components to reconstruct high-frequency details.

During the sampling process, the cascade texture is queried using

arbitrary UV coordinates. First, the UV LUT texture is accessed to obtain a new UV coordinate, and then this value is used to sample the color texture, as formulated below:

$$C(\mathbf{u}) = T_p(\mathcal{F}(T_c(\mathbf{u}))), \tag{7}$$

where $\mathbf{u}$ denotes the input UV coordinate, and $C(\mathbf{u})$ is the final lightmap color at the queried location. Here, both $T_c(\cdot)$ and $T_p(\cdot)$ represent texture lookups implemented with bilinear interpolation. The function $\mathcal{F}$ is a triangle-wave mapping, defined as:

$$\mathcal{F}(x) = 2 \cdot \left| \frac{x}{2} - \left\lfloor \frac{x}{2} + \frac{1}{2} \right\rfloor \right|. \tag{8}$$

The triangle-wave function ensures that coordinates remain within the range $[0,1]$, preserving differentiability and making it compatible with modern optimization techniques.

Importantly, the cascade texture representation is fully differentiable, which enables its integration into gradient-based optimization methods described in Section 3.3.3. This design achieves significant reduction in texture resolution while preserving high-frequency details and spatial coherence.

3.3.3. Differentiable Rendering Optimization

Notwithstanding the UV repacking process detailed in Section 3.3.1, interpolation artifacts may persist as discontinuities along triangle boundaries. To mitigate these seam artifacts, we employ a differentiable rendering-based optimization that jointly refines both texture components. Leveraging the Nvdiffrast differentiable renderer [LHK*20], we first initialize the textures: the color texture T_p is filled with the downsampled regenerated lightmap scaled by factor ϵ_p, while the UV LUT texture T_c adopts a standard 2D UV mapping scaled by ϵ_c.

During optimization, we rasterize the color texture onto the mesh from multiple stochastic viewpoints sampled across a unit sphere, producing screen-space radiance images. Losses are then computed directly in these rasterized images using a combination of L1 color fidelity and image-space gradient consistency. This formulation iteratively refines both textures, minimizing visual discontinuities along triangle boundaries while preserving the original illumination. The optimization converges to seamless, perceptually faithful reconstructions, despite substantial resolution reduction.

4. Experiments

4.1. Implementation Details

4.1.1. Technical Architecture

The proposed compression pipeline is implemented through a hybrid C++ / Python architecture. The core processing stages—from initial UV triangle standardization (Section 3.1) to UV repacking (Section 3.3.1)—are executed in C++. This module ingests mesh vertex coordinates, UV mappings, and HDR lightmaps, generating repacked lightmaps. The differentiable rendering optimization (Section 3.3.3) is implemented in Python using Nvdiffrast [LHK*20]. To accelerate triangle similarity assessment (Section 3.2.1), we leverage OpenMP [Ope18] for pairwise comparison parallelization. Triangles with radiance exceeding 2.0 are excluded from similarity analysis, as they are unlikely to recur and therefore do not warrant comparison. For the remaining triangles (pixel radiance $\in [0, 2.0]$), we normalize values to $[0,1]$ and quantize them into 8-bit integers, which reduces storage requirements and facilitates faster pixel-wise subtraction by lowering data precision.

4.1.2. Parameter setting

Key parameters are configured as follows: Similarity thresholds $\lambda = 0.03$ and $\lambda_h = 0.09$ control triangle grouping strictness; resolution scaling factors $\epsilon = 0.5$ (repacked lightmap), $\epsilon_p = 0.4$ (color texture), and $\epsilon_c = 0.7$ (UV LUT texture) balance compression ratios. Differentiable optimization employs Adam for 5,000 steps with mesh vertices normalized to $[-1,1]^3$. The learning rates are set to 5×10^{-4} for the color texture and 5×10^{-5} for the UV LUT texture. Camera positions are randomly sampled on a unit sphere (radius=3.0), and rendering occurs at 2048 × 2048 resolution to preserve high-frequency details during gradient-based refinement.

4.1.3. Dataset

Our dataset comprises 22 freely available mesh models curated from Sketchfab [Ske25], selected to be lightweight and geometrically clean, ensuring reliable UV unwrapping and effective clustering. The mesh models will be provided in the supplementary material. For each model, we generated lightmap UV coordinates using Blender [Ble25]. These assets were then integrated into a unified bedroom scene within Blender, where we established two physically based lighting configurations: a high-intensity daylight setup and a low-illuminance nighttime setup. Under each lighting condition, we baked 1024×1024 HDR lightmaps for every mesh, capturing global illumination effects. To ensure signal fidelity, all lightmaps underwent denoising via the NVIDIA OptiX denoiser [Rus19], resulting in a final corpus of 44 noise-reduced lightmaps suitable for compression analysis.

4.1.4. Evaluation Metrics

We employ three principal metrics to rigorously evaluate the efficacy of our compression framework. First, the compression ratio (CR) quantifies storage reduction through a bit-accurate comparison between compressed textures and the original lightmap. To address varying bit-depth requirements across texture formats, we define CR mathematically as:

$$CR = 1 - \frac{(W_p \times H_p \times \text{bpp}_p) + (W_c \times H_c \times \text{bpp}_c)}{W \times H \times \text{bpp}_{\text{orig}}}, \tag{9}$$

where $W \times H$ denotes the original lightmap dimensions, $\text{bpp}_{\text{orig}} = 48$ specifies the bits per pixel for our 3-channel 16-bit source data, where each channel is represented by a 16-bit half-precision floating point number. The compressed representation consists of two components: the color texture $\mathcal{T}_p$ with dimensions $W_p \times H_p$ and $\text{bpp}_p = 48$ bits per pixel (3-channel 16-bit format), alongside the UV LUT texture $\mathcal{T}_c$ with dimensions $W_c \times H_c$ and $\text{bpp}_c = 32$ bits per pixel (2-channel 16-bit coordinates). Second, perceptual fidelity is evaluated through render-space PSNR and 1-SSIM [WBSS04]. To ensure statistical robustness, we sample 50 camera viewpoints uniformly distributed on a unit sphere (radius=3.0), with all models normalized to a $[-1,1]^3$ bounding volume. For each view, the HDR radiance encoded in the lightmap is mapped onto the mesh

surface and rendered with Nvdiffrast [LHK*20] without additional shading. The resulting pixels directly represent radiance values, and metrics are averaged across all views to mitigate viewpoint bias.

4.2. Results

All experiments were executed on a desktop workstation equipped with an NVIDIA GeForce RTX 3060 GPU and Intel Core i7-10700 CPU. The hierarchical texture optimization leverages GPU-based differentiable rendering, while CPU-bound preprocessing stages (triangle standardization and similarity clustering) utilize multi-threaded parallelism.

Across both daylight and nighttime illumination scenarios, our compression framework achieves substantial storage reduction, with an average compression ratio (CR) of 83%, ranging from 75% to 91%, thereby consistently minimizing memory footprint. To assess visual fidelity, we compare the compressed and original HDR radiance maps by mapping them onto the mesh surface and rasterizing with Nvdiffrast. The resulting radiance images are quantitatively evaluated using PSNR and 1-SSIM, yielding average values of 51.20 and 0.0040, with minimum values of 34.66 and 0.0003, and maximum values of 68.60 and 0.0397, respectively. Figure 9 presents representative comparisons of rasterized HDR radiance maps, highlighting both the perceptual differences and the corresponding quantitative metrics. Figure 1 shows representative daylight renderings using the original and compressed lightmaps, where lightmaps provide the diffuse component and a uniform ambient term is added, without additional scene lighting.

4.3. Comparison

Table 1: *Comparison across different methods*

Method	CR (%) ↑	PSNR(dB) ↑	1-SSIM ↓
Ours	**83**	**51.79**	**0.0027**
DownSampling	**83**	51.10	0.0030
Knodt et al. [KG25]	**83**	39.05	0.0073
Luo et al. [LJP*23]	44	40.08	0.015

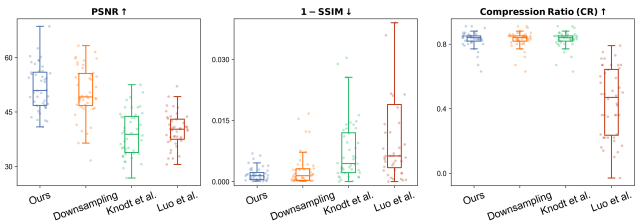


Figure 5: *Distribution of metrics across different methods. Each plot combines box plots with overlaid scatter points representing individual samples.*

We conduct comprehensive evaluations against simple bilinear DownSampling method and UV-based texture compression methods [LJP*23, KG25]. For equitable comparison, we implement critical experimental controls: DownSampling and Knodt et al. [KG25]'s methods are constrained to match our method's storage footprint by enforcing equivalent bit budgets, while Luo et al. [LJP*23]'s approach retains its native output resolution due to inherent architectural constraints.

Quantitative results in Table 1 and Figure 5 reveal our consistent improvements over prior methods across both PSNR and SSIM, indicating enhanced rendering fidelity. Qualitative evaluations in Figure 6 provide deeper visual insights: simple DownSampling introduces pervasive blurring artifacts that degrade high-frequency illumination details, particularly evident in shadow boundaries; in contrast, our selective region compression preserves sharp edges and maintains illumination integrity. We apply the seam carving component of Knodt et al. [KG25] to reduce texture resolution, but observe severe color bleeding under high compression. In densely packed lightmap UVs, seams often traverse boundaries between adjacent charts, and their removal merges distinct lighting responses, causing light to leak across regions. The other component of the same framework—symmetric chart merging—is largely inapplicable, as lightmaps rarely exhibit mirror-symmetric illumination due to view- or environment-dependent lighting. Luo et al. [LJP*23]'s monochromatic triangle matching also yields limited gain due to the scarcity of uniform-color regions in gradient-rich lightmaps. These results highlight the incompatibility of general UV simplification strategies with lightmap compression, validating our specialized approach.

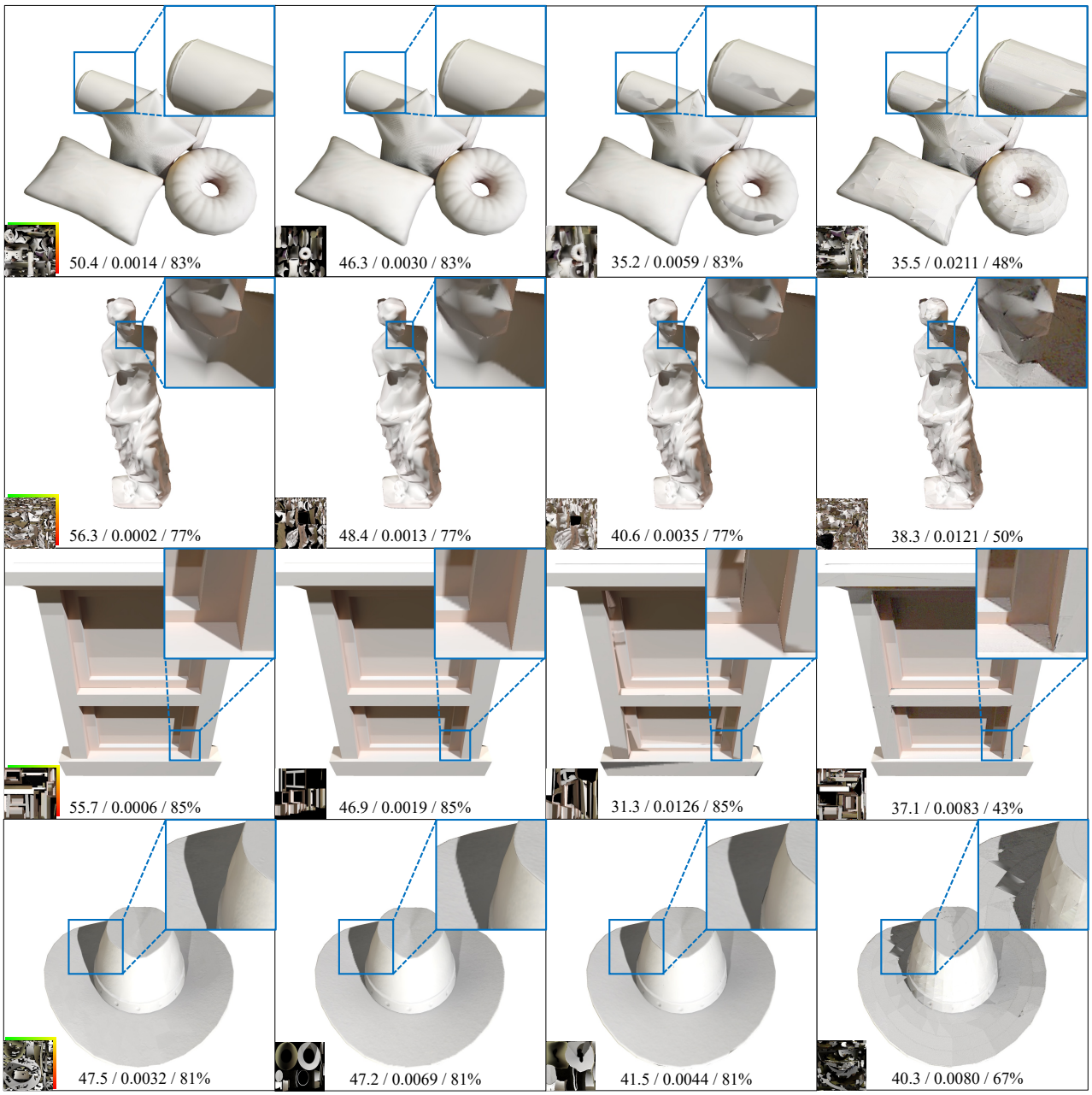
In terms of compression runtime efficiency, our lightmap compression algorithm requires an average of **156 seconds** per lightmap, compared to **258 seconds** for Luo et al. [LJP*23] and **206 seconds** for Knodt et al. [KG25]. Conventional downsampling remains the fastest at under one second. At render time, the additional UV LUT lookup incurs negligible overhead (adding less than 10 µs per frame in our WebGL implementation), demonstrating minimal impact on real-time performance.

4.4. Compatibility with Block-based Compression

Table 2: *Performance of ASTC-compatible compression.*

Method	CR (%) ↑	PSNR (dB) ↑	1-SSIM ↓
Ours	83	**51.79**	**0.0027**
Ours+ASTC	**87**	49.58	0.0029

We demonstrate that our framework is fully compatible with conventional block-based texture compression schemes. Specifically, applying ASTC with a 4×4 block size to the color texture generated by our method yields a hybrid compression approach with a higher overall compression ratio. The additional compression gain arises from the reduced bits per pixel (bpp) in the color texture after ASTC encoding, which reduces the bpp_p from 48 to 6, directly contributing to lower storage requirements. As shown in Table 2, this combination provides a more aggressive compression option, with only a modest impact on perceptual fidelity, offering greater flexibility when prioritizing storage efficiency.



(a) Ours (b) DownSampling (c) Knodt et al. (d) Luo et al.

Figure 6: *Qualitative comparison. Compared with direct lightmap downsampling, our method produces smoother hard shadow boundaries without introducing blocky artifacts. In contrast to prior UV-based texture optimization approaches [LJP*23, KG25], our framework avoids color bleeding artifacts and better preserves shading consistency across surfaces. Quantitative metrics (PSNR / 1–SSIM / Compression Ratio) are provided below each image for reference. The visual quality metrics are averaged over all viewing directions.*

4.5. Ablation Study

4.5.1. Compression parameters

We conduct a systematic evaluation of three critical parameters governing our compression pipeline: the repacked lightmap scaling factor (ϵ), color texture resolution ratio (ϵ_p), and UV LUT resolution ratio (ϵ_c). Through controlled experiments where each parameter was varied individually while holding the other two at their identified optimal values, we identify precise performance thresh-

olds essential for balancing compression efficiency and visual fidelity.

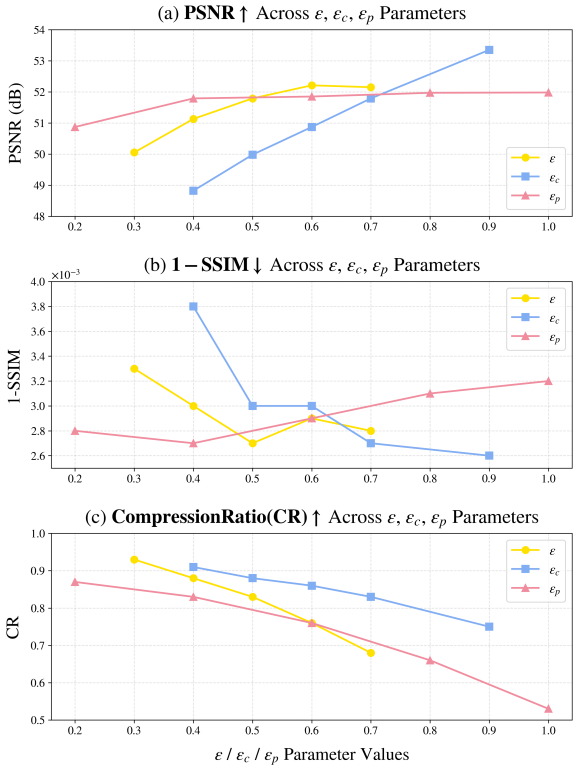


Figure 7: *Impact of different parameter settings.*

As shown in Figure 7, our analysis reveals that reducing ϵ below 0.5 triggers significant quality degradation due to progressive texture detail loss, while ϵ_p values under 0.4 cause abrupt reductions in color variation richness. For ϵ_c, we observe that settings below 0.7 substantially impair high-frequency detail preservation. After comprehensive testing across diverse scenes, we establish the optimal configuration as $\epsilon = 0.5$, $\epsilon_p = 0.4$, and $\epsilon_c = 0.7$. Notably, higher ϵ_p values yield diminishing quality returns at the expense of compression gains, whereas lower ϵ_c values disproportionately degrade high-frequency features critical for illumination accuracy.

4.5.2. Compression Strategy

We conduct ablation studies on two core components: 1) UV repacking (Section 3.3.1) and 2) cascade texture representation (Section 3.3.2).

UV repacking. We bypassed the UV repacking stage while keeping the resolution scaling factors ϵ_c and ϵ_p fixed, and initialized the cascade texture directly from the original lightmap rather than the repacked version to isolate the effect of UV repacking. Table 3 and Figure 8 indicate that omitting UV repacking slightly improves the fidelity of the rasterized HDR radiance maps due to the preservation of more UV information, but substantially reduces the compression ratio compared to the full pipeline.

Cascade texture. We remove the cascade texture module entirely and use the repacked lightmap directly as output, without performing gradient-based optimization. As reported in Table 3, UV repacking has a greater impact on compression ratio than the cascade texture. However, the absence of optimization prevents effective correction of seams and jagged edges, resulting in decreased rasterized HDR radiance fidelity, as illustrated in Figure 8 (right).

These findings demonstrate that the combined application of UV repacking and the cascade texture representation is essential for achieving high-fidelity lightmap compression, effectively balancing compression efficiency and rasterized HDR radiance quality.

Table 3: *Ablation study results on lightmap compression.*

Method	CR (%) ↑	PSNR (dB) ↑	1-SSIM ↓
Ours	**83**	51.20	0.0040
w/o UV repacking	51	**51.93**	**0.0021**
w/o cascade texture	64	39.88	0.0070

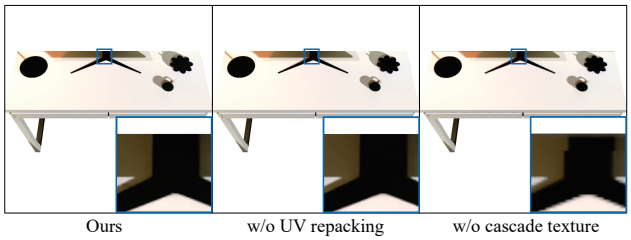


Figure 8: *Visual comparison of ablation experiments. Both UV repacking and cascade texture contribute to improved compression ratio and rasterized HDR radiance fidelity*

5. Limitations and Conclusions

5.1. Limitations

Our method struggles with sparsely triangulated surfaces exhibiting complex intra-triangle illumination gradients, such as large planar surfaces in architectural scenes with intricate shadow transitions. In these cases, high color variance within individual triangles reduces compression efficiency, diminishing our advantage over naive downsampling. Future work may explore adaptive subdivision based on gradient magnitude and extend the hierarchical framework to pixel-level compression within triangles for improved granularity.

5.2. Conclusions

We present the first dedicated framework for lightmap texture compression, fully compatible with modern real-time rendering pipelines. Our method exploits color-coherent UV clustering to identify and group perceptually similar triangles, enabling efficient UV repacking and reduced redundancy. Additionally, by integrating a cascade texture representation into a differentiable rendering pipeline, we achieve substantial resolution reduction while preserving high visual fidelity and minimizing seam artifacts. These contributions provide a practical and scalable solution to the memory demands of lightmaps, supporting more visually rich and expansive virtual environments.

Acknowledgments

Xiaogang Jin was supported by the National Natural Science Foundation of China (Grants: U25A20440, 62472373).

References

[AMTM23] AGUSTSSON E., MINNEN D., TODERICI G., MENTZER F.: Multi-realism image compression with a conditional generator. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2023), pp. 22324–22333. 2

[ATM*19] AGUSTSSON E., TSCHANNEN M., MENTZER F., TIMOFTE R., VAN GOOL L.: Generative adversarial networks for extreme learned image compression. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 221–231. 2

[AvAB*19] ALAKUIJALA J., VAN ASSELDONK R., BOUKORTT S., BRUSE M., COMȘA I.-M., FIRSCHING M., FISCHBACHER T., KLIUCHNIKOV E., GOMEZ S., OBRYK R., POTEMPA K., RHATUSHNYAK A., SNEYERS J., SZABADKA Z., VANDEVENNE L., VERSARI L., WASSENBERG J.: JPEG XL next-generation image compression architecture and coding tools. In *Applications of Digital Image Processing XLII* (2019), vol. 11137, SPIE, p. 111370K. 2

[Ble25] BLENDER FOUNDATION: Blender, 2025. URL: https://www.blender.org. 6

[BMS*18] BALLÉ J., MINNEN D., SINGH S., HWANG S. J., JOHNSTON N.: Variational image compression with a scale hyperprior. In *International Conference on Learning Representations* (2018). 2

[CMH*18] CHEN Y., MURHERJEE D., HAN J., GRANGE A., XU Y., LIU Z., PARKER S., CHEN C., SU H., JOSHI U., CHIANG C.-H., WANG Y., WILKINS P., BANKOSKI J., TRUDEAU L., EGGE N., VALIN J.-M., DAVIES T., MIDTSKOGEN S., NORKIN A., DE RIVAZ P.: An overview of core coding tools in the av1 video codec. In *2018 Picture Coding Symposium (PCS)* (2018), pp. 41–45. 2

[CWH93] COHEN M. F., WALLACE J., HANRAHAN P.: *Radiosity and realistic image synthesis*. Academic Press Professional, Inc., 1993. 1

[DGA*21] DUPONT E., GOLIŃSKI A., ALIZADEH M., TEH Y. W., DOUCET A.: Coin: Compression with implicit neural representations, 2021. URL: https://arxiv.org/abs/2103.03123, arXiv:2103.03123. 2

[DM79] DELP E., MITCHELL O.: Image compression using block truncation coding. *IEEE Transactions on Communications 27*, 9 (1979), 1335–1342. 3

[DMD*23] DATTA S., MARSHALL C., DONG Z., LI Z., NOWROUZEZAHRAI D.: Efficient graphics representation with differentiable indirection. In *SIGGRAPH Asia 2023 Conference Papers* (2023), SA '23, Association for Computing Machinery. 3, 5

[Fen03] FENNEY S.: Texture compression using low-frequency signal modulation. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2003), HWWS '03, Eurographics Association, p. 84–91. 3

[FHL*24] FARHADZADEH F., HOU Q., LE H., SAID A., RAUWENDAAL R., BOURD A., PORIKLI F.: Neural graphics texture compression supporting random access. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XXXVII* (2024), Springer-Verlag, p. 412–429. 3

[GPW*23] GHOUSE N. F., PETERSEN J., WIGGERS A., XU T., SAUTIÈRE G.: A residual diffusion model for high perceptual quality codec augmentation, 2023. URL: https://arxiv.org/abs/2301.05489, arXiv:2301.05489. 2

[HYP*22] HE D., YANG Z., PENG W., MA R., QIN H., WANG Y.: Elic: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2022), pp. 5708–5717. 2

[KG25] KNODT J., GAO X.: Texture size reduction through symmetric overlap and texture carving. *ACM Transactions on Graphics 44*, 1 (2025). 2, 3, 7, 8

[LHK*20] LAINE S., HELLSTEN J., KARRAS T., SEOL Y., LEHTINEN J., AILA T.: Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics 39*, 6 (2020). 6, 7

[LJP*23] LUO Y., JIN X., PAN Z., WU K., KOU Q., YANG X., GAO X.: Texture atlas compression based on repeated content removal. In *SIGGRAPH Asia 2023 Conference Papers* (2023), SA '23, Association for Computing Machinery. 2, 3, 7, 8

[MENU*23] MUCKLEY M., EL-NOUBY A., ULLRICH K., JÉGOU H., VERBEEK J.: Improving statistical fidelity for neural image compression with implicit local likelihood models. In *Proceedings of the 40th International Conference on Machine Learning* (2023), ICML'23, JMLR.org. 2

[Mic20] MICROSOFT CORPORATION: Texture block compression in direct3d 11, 2020. Accessed September 12, 2025. URL: https://learn.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11. 2, 3

[NLP*12] NYSTAD J., LASSEN A., POMIANOWSKI A., ELLIS S., OLSON T.: Adaptive scalable texture compression. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (2012), EGGH-HPG'12, Eurographics Association, p. 105–114. 2, 3

[Ope18] OPENMP ARCHITECTURE REVIEW BOARD: Openmp application program interface version 5.0, 2018. Available at: https://www.openmp.org/specifications/. 6

[Rus19] RUSSELL D.: Nvidia ai denoiser command line tool, 2019. Accessed: 2025-09-13. URL: https://github.com/DeclanRussell/NvidiaAIDenoiser. 6

[SAM05] STRÖM J., AKENINE-MÖLLER T.: ipackman: high-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (2005), HWWS '05, Association for Computing Machinery, p. 63–70. 3

[Ske25] SKETCHFAB: Sketchfab: Platform for publishing and sharing 3d content, 2025. Accessed: 2025-09-13. URL: https://sketchfab.com. 6

[SP07] STRÖM J., PETTERSSON M.: Etc 2: texture compression using invalid combinations. In *Graphics Hardware* (2007), vol. 7, pp. 49–54. 2, 3

[SPY*22] STRÜMPLER Y., POSTELS J., YANG R., GOOL L. V., TOMBARI F.: Implicit neural representations for image compression. In *Computer Vision – ECCV 2022: 17th European Conference, October 23–27, 2022, Proceedings, Part XXVI* (2022), Springer-Verlag, pp. 74–91. 2

[VSW*23] VAIDYANATHAN K., SALVI M., WRONSKI B., AKENINE-MOLLER T., EBELIN P., LEFOHN A.: Random-access neural compression of material textures. *ACM Transactions on Graphics 42*, 4 (2023). 3

[Wal92] WALLACE G.: The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics 38*, 1 (1992), xviii–xxxiv. 2

[WBSS04] WANG Z., BOVIK A., SHEIKH H., SIMONCELLI E.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612. 6

[Wik25] WIKIPEDIA CONTRIBUTORS: S3 texture compression, 2025. [Online; accessed 05-September-2025]. URL: https://en.wikipedia.org/wiki/S3_Texture_Compression. 3

[You17] YOUNG J.: xatlas: Mesh parameterization / uv unwrapping library, 2017. URL: https://github.com/jpcy/xatlas. 5

Figure 9: *Visual results. For each mesh, we present two separately baked lightmaps, each visualized with three components: the rasterized HDR radiance map from the original lightmap (left), the rasterized HDR radiance map from the compressed textures (middle), and the amplified visual difference between them (right). Quantitative metrics (PSNR/1-SSIM/CR) are provided on the visual difference images. The visual quality metrics are averaged over all viewing directions.*