



BACHELORARBEIT

REAL-TIME IMAGE-BASED LIGHTING WITH IRRADIANCE CUBEMAPS FROM 3D GAUSSIAN SPLATTING ENVIRONMENTS

Verfasser

Philipp Orozco-Carazo

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, Jänner 2026

Studienkennzahl lt. Studienblatt: 033 521

Fachrichtung: Informatik - Allgemein

Betreuerin / Betreuer: Univ.-Prof. Dr. Helmut Hlavacs

Contents

1	Introduction	3
2	Related Work	4
3	Technical Foundations	8
3.1	Conceptual Dichotomies	8
3.2	Fundamental Concepts	8
3.3	Techniques and Methods	10
3.4	Technologies and Specifications	11
4	Thesis Contribution	12
4.1	Gaussian Splatting Renderer	12
4.2	Diffuse Irradiance Cubemap Generator	13
4.3	Gaussian Splatting Cubemap Lighting Demo	13
4.4	Limitations	14
4.5	Usage of Generative AI	15
5	Evaluation	15
5.1	Benchmark Setup	16
5.2	Evaluation Results	17
5.3	Discussion	18
6	Conclusion	20

Abstract

Gaussian Splatting is a volume rendering technique used in 3D Computer Graphics (3D CG). The recent contribution 3D Gaussian Splatting (3DGS) leveraged modern machine learning approaches to create real-time radiance field renderings of photorealistic scenes. In comparison to Gaussian Splatting volumetric renderings, traditional objects in 3D CG are usually defined by mesh-based geometry and can be made to dynamically interact with lighting in a scene through Physically Based Rendering (PBR). The goal of the thesis project is to create a demonstration application inside the Vienna Vulkan Engine (VVE) [16] codebase. The thesis implementation renders a Gaussian Splatting environment and a mesh-based object together in a scene using an Image-Based Lighting (IBL) approach to illuminate the object with diffuse ambient lighting, sampled from an irradiance cubemap, which is generated by light probing the surrounding Gaussian environment from the objects position in real-time. Evaluation of technical benchmarks demonstrates the feasibility of this approach but also confirms its limitations.

1 Introduction

In the field of 3D Computer Graphics (3D CG), combining both traditional mesh-based 3D objects using Physically Based Rendering (PBR) [7] and 3D Gaussian Splatting (3DGS) scenes into a single rendering pipeline and having the PBR shader interact with the scene lighting poses conceptual and technical challenges. The most fundamental limitation when trying to generalize this approach is that it is only by convention that some 3DGS objects can be considered 3D environments. In the scope of this thesis, the term *3DGS environment* is used to refer to 3DGS scenes that have all their intended viewpoints inside of the point cloud, i.e., for every view position, the viewer is predominantly enclosed by Gaussian splats. Within the scope of this thesis, it is assumed that the 3DGS environments used are full spherical captures of scenes, i.e., there are no “gaps” or “holes”. Without this assumption, reconstruction methods like inpainting would have to be considered.

Furthermore, when trying to employ Image-Based Lighting (IBL) to recreate realistic ambient light, the Spherical Harmonics (SH), which store view-dependent radiance information for each Gaussian kernel, can only be leveraged for diffuse lighting, but not for specular highlights [40]. SH can be considered a type of Fourier series; they therefore suffer the same inherent limitation when trying to represent high-frequency information (e.g., Gibbs ringing artifacts [33], see Fig. 1, second from the left).



Figure 1: From left: Ground Truth, Spherical Harmonics (displaying Gibbs ringing artifacts), Spherical Gaussians, Spherical Voronoi, Voronoi Cells. From work by Di Sario et al. [33]

When considering realistic environment scenes with static lighting, the fidelity of some Gaussian Splatting environments greatly surpasses what is currently possible with conventional rendering techniques [22]. In order to leverage this fidelity in real-time digital interactive media such as video games and Virtual Reality (VR) applications, the current state

of technology necessitates a hybrid rendering approach, as most 3D assets in conventional 3D CG applications are mesh-based, designed to be compatible with physics or animation requirements and optimized for efficient rendering. Furthermore, considering the fact that most related work leverages Machine Learning (ML) methods and the increasing environmental impact of ML-based technologies, the thesis explores the feasibility and viability of traditional PBR methods and elaborates on potential enhancements to the hybrid rendering pipeline.

The focus of this thesis is to demonstrate that, given a high-performance 3DGS renderer, real-time IBL via diffuse irradiance cubemaps obtained using light probing is feasible and has further potential to be optimized. The thesis also explores the performance impact of scaling factors and elaborates on future enhancements. The thesis implementation goals were to integrate an existing high-performance Gaussian Splatting renderer into the *Vienna Vulkan Engine (VVE)*, implement a light probe pipeline in VVE and create an environmental cubemap image, which is then filtered to create an irradiance map. The filter can convolve the image using hemisphere sampling or apply a simple box filter. When calculating the diffuse ambient light term in the PBR shader, the contribution of the sampled irradiance map is added to the result. Since implementing an optimized Gaussian Splatting renderer from scratch was out of the project’s scope, the rendering library *vkgs* [29] written in Vulkan was the starting point for the thesis project. The thesis implementation does not introduce any fundamental changes to existing rendering approaches and does not add additional machine learning aspects to the 3DGS pipeline. For the sake of simplicity, advanced or optional features in *vkgs* like special synchronization structures, i.e., fences and semaphores, have not been transferred to the adapted implementation. Didactic goals were familiarization with the concepts of lighting techniques in 3D CG, Neural Radiance Field (NeRF) and 3DGS, the Vulkan API and Vulkan render optimization techniques, and to gain experience in merging codebases of different origin by adapting the existing *vkgs* codebase and integrating it into the existing educational codebase of VVE, while also making an original contribution by demonstrating a runtime rendering and composition technique, which involves extracting lighting information from a Gaussian Splatting scene to illuminate traditional mesh objects with diffuse light sampled from a generated irradiance cubemap.

The thesis first surveys related work on Gaussian Splatting, inverse rendering and lighting techniques, then introduces the technical foundations including rendering concepts, methods and technologies used. It describes the thesis contribution comprising the Gaussian Splatting renderer, irradiance cubemap generator and demonstration application, presents the evaluation methodology and benchmark results, and concludes with a summary and future work. The thesis addresses two research questions: (*Q1*) Is it possible to extract usable ambient lighting information from an unmodified 3DGS environment via light probing and irradiance cubemap generation while maintaining real-time Frames per Second (FPS)? (*Q2*) What is the performance impact of light probing and irradiance cubemap generation on real-time rendering, and what are the architectural challenges when integrating a Gaussian Splatting renderer into an existing Vulkan engine codebase?

2 Related Work

Westover [46] introduces splatting as a volume rendering technique that projects each 3D sample onto the image plane using a reconstruction kernel instead of casting rays. 3DGS [22] builds directly on this, using Gaussian kernels, tile-based sorting, and alpha blending for front-to-back compositing.

Polygon File Format (PLY) stores 3D objects as vertices, faces, and other elements with attached properties, available in ASCII and binary formats [42]. Its key strength is extensibility: applications can add custom properties to any element, and programs that don't understand them simply skip them. 3DGS [22] uses PLY files to store trained models, extending the vertex element with properties for position, spherical harmonic coefficients, opacity, scale, and rotation.

Structure-from-Motion (SfM) [34, 35] reconstructs camera poses and 3D point clouds from photos, the starting point for training 3DGS [22, 28]. GLOMAP jointly optimizes camera positions and 3D points in a single step using camera ray constraints, unlike traditional methods that solve rotations, positions, and triangulation separately [28].

NeRF represents a 3D scene as a neural network that maps any position and viewing direction to a color and density, rendered by sampling points along camera rays [27]. Training takes 1–2 days per scene and the method could not render in real time, motivating 3DGS [22] which uses explicit primitives for real-time performance.

3D Gaussian Splatting represents scenes as millions of 3D Gaussians with position, covariance, opacity, and Spherical Harmonics color, rendered via tile-based sorting and alpha blending [22]. An optimization procedure starting from SfM point clouds uses adaptive density control to add, split, and remove Gaussians, achieving 30+ FPS at 1080p. Because of the implications for real-time applications, the work had a strong impact on the 3D CG industry and the scientific community.

Wang et al. [45] replace the 48 SH coefficients in 3DGS with Spherical Gaussians, which are Gaussian functions defined on the surface of a sphere that each represent a directional color lobe. A sharpness parameter controls the angular width of each lobe, allowing compact representation of both sharp specular and broad diffuse appearance.

Song et al. [38] assign each 2D Gaussian surfel an optimizable texture map sized proportionally to its scale, disentangling geometry from appearance so that a single surfel can represent spatially varying detail like fabric or text. Per-ray depth sorting replaces the standard global per-view ordering to eliminate popping artifacts that become severe with per-surfel textures, and a frustum-based sampling method reduces aliasing at lower resolutions.

Ramamoorthi and Hanrahan [31] prove that only 9 Spherical Harmonics coefficients (order $l \leq 2$) are needed for diffuse irradiance with under 1% average error. This justifies the use of low-resolution irradiance maps (32×32) and explains why multiple papers note that “SH can only represent low-frequency lighting.”

Debevec [10] defines the IBL pipeline: capture omnidirectional High Dynamic Range (HDR) light, map it onto an environment sphere, and simulate how it illuminates objects. Light probes must be omnidirectional and high dynamic range, since standard photos encode values nonlinearly and saturate bright areas.

Xu et al. [49] approximate large convolution kernels using weighted box-filtered mipmaps, where computational cost depends only on image resolution, not kernel size. Irradiance map generation takes 0.386 ms compared to over 30 minutes for ground truth. This was the thesis motivation to explore the approach of box-filtered downsampling (512×512 to 32×32) instead of proper cosine convolution for irradiance maps.

De Vries [9] describes a practical IBL implementation that convolves an environment cubemap into a diffuse irradiance map using cosine-weighted hemisphere sampling. The thesis convolution filter implementation is based on this approach.

SuGaR regularizes 3DGS to encourage flat, surface-aligned Gaussians, then extracts meshes via Poisson reconstruction [14]. Gaussians are bound to mesh triangles for editing while maintaining rendering quality.

Building on SuGaR [14], Frosting adds a layer of Gaussians around the extracted mesh surface with adaptive thickness [15]. When the mesh is deformed, all Gaussians automatically adjust position, rotation, and scale.

GaMeS parameterizes each Gaussian by mesh triangle vertices, so position, rotation, and scale update automatically when the mesh is deformed [44]. Without an input mesh, flat Gaussians are converted into a “Triangle Soup” of editable unconnected triangles. Quality is comparable to vanilla 3DGS on NeRF-Synthetic and MipNeRF-360 [2] while enabling real-time editing. MeshSplats [41] later noted that this flat Gaussian approach naturally fits mesh conversion.

MeshSplats approximates each flat Gaussian as a polygon of 8 triangles, converting trained 3DGS scenes into meshes renderable in Blender and Nvdiffrast [41]. An optimization step refines vertices, removes artifacts, and recovers fine details. The method works best with flat Gaussians (2DGS [17], GaMeS [44]) and enables ray tracing for shadows and reflections.

Huang et al. [17] replace 3D Gaussians with flat 2D Gaussian disks embedded in 3D space, where the surface normal is inherently defined by each disk’s tangent plane. A ray-splat intersection replaces the affine projection approximation used in 3DGS [22], improving accuracy at oblique viewing angles. Depth distortion and normal consistency regularization losses further encourage splats to concentrate on and align with actual surfaces.

DN-Splatter adds depth and normal supervision during 3DGS training using sensor readings or monocular networks, forcing Gaussians into flat shapes aligned with surfaces [43]. This enables direct mesh extraction via Poisson reconstruction, outperforming SuGaR [14] and 2DGS [17] on indoor datasets. The results confirm that without geometric priors, Gaussian methods struggle in textureless regions.

Dai et al. [8] flatten 3DGS into 2D ellipses (Gaussian surfels) by setting the z-scale to zero, making the z-axis the unambiguous surface normal. Meshes are extracted via Poisson reconstruction from rendered depth and normal maps.

Jin et al. [19] jointly reconstruct a radiance field and a physically-based model (normals, materials, lighting) using a tensor-factorized scene representation. Unlike previous NeRF-based methods that require costly pre-computation, the efficient representation allows direct ray marching for shadows and indirect illumination during training.

NeRV replaces NeRF’s [27] color output with BRDF parameters, enabling relighting under new conditions [39]. A “Neural Visibility Field” learns to approximate visibility cheaply, reducing direct lighting cost from quadratic to linear. Later 3DGS-based methods (R3DG [11], GS-IR [25], GIR [36], GI-GS [6]) build on similar material decomposition with explicit Gaussians.

Gao et al. [11] extend each Gaussian with normal vectors, BRDF parameters, and indirect lighting components, using Bounding Volume Hierarchy (BVH) ray tracing for shadows. Indirect lighting is learned for a specific scene and becomes invalid when objects move to new environments. R3DG requires retraining to decompose materials and lighting. The thesis implementation’s real-time light probe approach tries to address this limitation.

GS-IR decomposes 3DGS scenes into geometry, materials, and environment lighting, using depth-gradient regularization for normals and baked SH occlusion volumes for indirect illumination [25]. Rendering uses the split-sum IBL approach. The authors note SH can only represent low-frequency lighting, limiting the method to diffuse indirect illumination.

GIR decomposes 3DGS scenes into geometry, materials, and lighting, using “directional masking” for normal estimation and a neural network for HDR environment maps [36]. Grid-based ray tracing with voxels handles indirect illumination. Unlike GS-IR [25] which bakes occlusion, GIR traces rays at runtime.

GI-GS combines deferred shading with path tracing on G-buffers, using ray marching on the depth map for occlusion and indirect lighting [6]. It renders six cubemap faces to capture global geometry beyond the camera frustum. Results improve over GS-IR [25] especially in indoor scenes, though environment maps “may fail to capture spatially varying lighting.”

Wu et al. [48] use deferred shading to avoid blending artifacts from forward shading in 3DGS relighting, rasterizing attributes into G-buffers before per-pixel shading. The normal field is obtained from a jointly-trained Signed Distance Field (SDF) network, and lighting uses a $6 \times 512 \times 512$ HDR cubemap with split-sum approximation.

Ye et al. [51] use deferred shading to render specular reflections with 3DGS. A splatting pass produces screen-space normal and reflection strength maps, followed by a per-pixel environment map query for specular color. The method does not perform full material decomposition and leaves diffuse and rough reflections to base SH colors.

Yao et al. [50] extend 3DGS-DR [51] with full Disney Bidirectional Reflectance Distribution Function (BRDF) properties (albedo, metallic, roughness) and inter-reflection via ray tracing on periodically extracted meshes for visibility. The method uses 2DGS [17] as its base representation and split-sum approximation for efficient specular rendering.

ReCap addresses the albedo-lighting ambiguity by optimizing multiple environment maps from cross-environment captures that share common material attributes [24]. The method removes the metallic parameter from the split-sum shading function and constrains learned lighting to linear HDR space for direct use of standard HDR maps during relighting.

Teuber et al. [40] optimize light probe placement in 3DGS environments via geometry-based iterative merging, starting with equally spaced probes and repeatedly removing redundant ones while placing probes on object surfaces. Indoor environments with occlusions require more probes, while outdoor scenes work with fewer.

Di Sario et al. [33] introduce Spherical Voronoi (SV) functions as an alternative to SH for light probes, avoiding Gibbs ringing artifacts at sharp lighting discontinuities. Learnable probes throughout 3DGS scenes approximate spatially varying lighting via interpolation, significantly improving reflection appearance over SH. The paper notes that “relying exclusively on far-field illumination is an incorrect model when glossy objects are in proximity of other objects,” confirming the thesis limitation of single stationary probes.

Skorokhodov et al. [37] leverage diffusion models to enforce lighting consistency during insertion of a 3DGS object into a 3DGS scene. R3DG [11] struggles to distinguish dark colors from shadows, and its learned lighting becomes invalid in new scenes.

MVInpainter generates consistent inpainting across multiple viewpoints simultaneously using a video diffusion model, avoiding the blurry results of SDS and the incomplete shapes of single-view methods [52]. Testing on MipNeRF-360 [2] scenes shows better quality than previous methods. Like D3DR [37], this inserts objects into Gaussian scenes.

Bolduc et al. [4] convert LDR photos to HDR using diffusion models, then fit a 3DGS representation usable as light emitters in renderers like Blender. The paper compares lighting representations, noting that environment maps (IBL) provide high-frequency detail but “cannot yield near-field lighting effects.”

Liu et al. [26] remove objects from 3DGS scenes by inpainting color with Stable Diffusion XL and depth with a specialized diffusion model, then unprojecting back to 3D. The method completes in 40 seconds compared to 5 hours for previous approaches. Like D3DR [37] and MVInpainter [52], this modifies 3DGS scenes using diffusion models.

3 Technical Foundations

This section introduces the concepts, methods and technologies that the thesis builds on. It provides a framework for classifying related research, explains the rendering principles underlying the hybrid pipeline, and covers the methods employed in the thesis from radiance fields to image-based lighting. The section also covers the Vulkan API, the external vkgs renderer that served as the basis for the Gaussian Splatting implementation, and the supporting libraries used in the thesis project.

3.1 Conceptual Dichotomies

In order to help categorize related work and the thesis project, dichotomous concepts are used for classification. In 3D CG, illumination is commonly separated into *direct* and *indirect* light. Direct light refers to light that travels from a source to a surface without interaction, while indirect light refers to light that has been reflected, refracted or scattered by other surfaces in the scene before reaching the target surface. A related distinction is between local and global illumination, where local illumination only considers the direct relationship between light sources and surfaces, while global illumination accounts for all light transport in a scene including indirect contributions. Furthermore, the reflective behavior of a surface can be separated into *diffuse* and *specular* components [7]. Diffuse reflection scatters light uniformly in all directions regardless of the viewing angle, while specular reflection is view-dependent and produces highlights concentrated around the mirror reflection direction.

The light itself can be measured as *radiance*, the light in a single direction, or *irradiance*, the total light reaching a surface point from all directions. The thesis converts a radiance cubemap into an irradiance map so that each surface normal can look up its total incoming diffuse light. *Spherical Harmonics* are well suited to represent low-frequency diffuse illumination, while environment maps [3] are typically used to capture the full directional radiance of a scene including high-frequency specular information [33, 22, 40]. In the context of rendering architectures, *forward rendering* computes all lighting for each fragment during rasterization, while *deferred rendering* first writes surface attributes to geometry buffers (G-buffers) [32] and then computes lighting in a separate full-screen pass, which is more efficient for scenes with many light sources. Gaussian Splatting and mesh-based rendering represent two fundamentally different scene representations: Gaussian Splatting uses volumetric primitives with learned appearance properties, while mesh-based rendering uses explicit surface geometry with material parameters that interact with lighting through the BRDF.

A further distinction exists within 3DGS between environment scenes and isolated objects. Standard 3DGS training learns all visible content in the input images, including the background. Isolated 3DGS objects are produced by training with masked input images, where the background is removed so that the optimization only learns the foreground object. The result is a point cloud that represents a single object without its surroundings, similar in concept to a traditional 3D mesh asset. This thesis focuses on 3DGS environments, where the viewer is enclosed by the Gaussian splats, as opposed to isolated objects which would be placed inside a scene.

3.2 Fundamental Concepts

The thesis builds on well-established ideas and concepts in 3D CG. *Splatting* is a family of rendering techniques in 3D CG that project point-like primitives onto the screen and accumu-

late their contributions [46]. The primitives can take different forms such as discs, ellipsoids or Gaussians, and the technique is used in volume rendering, point cloud visualization and particle systems. Gaussian Splatting is a specific variant that defines each primitive as a Gaussian kernel. In 3DGS, view-dependent color information is stored per Gaussian using *Spherical Harmonics*, which are the spherical equivalent of a Fourier series. Each Gaussian ellipsoid stores degree-3 SH coefficients that encode how its appearance changes with viewing direction. As a frequency-based representation, SH can represent smooth color variations but struggle with sharp transitions. This limitation is known as the *Gibbs phenomenon* [12]: when a signal with sharp discontinuities is represented by a finite number of Fourier terms, the truncated series produces oscillating overshoot and undershoot near the discontinuity that does not disappear as more terms are added. In the context of 3DGS, this means that SH cannot faithfully represent specular highlights or sharp lighting boundaries, which is why the thesis uses cubemap-based IBL instead of directly reading SH coefficients for lighting (see Fig. 1).

To combine overlapping layers, a method called *alpha compositing* [30] is used. Each pixel has an alpha value that ranges from 0 (fully transparent) to 1 (fully opaque). The most common operation is “over”, where the foreground color is blended with the background according to its alpha. When multiple semi-transparent layers are stacked, each layer blocks a portion of the light from the layers behind it, which is tracked as transmittance. In Gaussian Splatting, each splat has an opacity parameter that acts as its alpha value, and splats must be sorted by depth so that the blending produces a correct result. If splats are composited in the wrong order, visible artifacts occur.

In *rasterization*, geometry is projected onto the screen and converted into fragments, which are then shaded to produce pixel colors. Instead of computing lighting during this step, *deferred rendering* splits the process into two passes [32]. The first pass renders all geometry but only stores surface properties such as position, normal, albedo and roughness into screen-sized textures called geometry buffers (G-buffers), without computing any lighting. The second pass reads these G-buffers and computes lighting per pixel in a full-screen pass, so that lighting is calculated only once per visible surface point regardless of scene complexity. In the thesis, the deferred renderer handles the mesh object: it writes surface properties to G-buffers, and the lighting pass samples the irradiance cubemap using the stored normal to add the ambient contribution.

Environment mapping [3] is a technique that captures the surrounding scene as a texture, which can then be sampled during rendering to simulate reflections or lighting from the environment. A *cubemap* [13] is a specific form of environment map that stores the scene as six square images, each representing a face of a cube centered at the capture point, with each face covering a 90-degree field of view. Together the six faces cover the full sphere of directions around the capture point. Cubemaps are widely used in real-time rendering for reflections, skyboxes and IBL. In the thesis, the Gaussian environment is rendered to the six faces of a cubemap, which is then filtered to produce an irradiance map for diffuse ambient lighting.

To produce a usable irradiance map from an environment cubemap, the map is filtered using *convolution*, a mathematical operation that computes a weighted average at each point of a function using a second function called the kernel [9]. In IBL, the environment cubemap is convolved by sampling light directions over the hemisphere above each surface point, with each sample weighted by $\cos(\theta)$ according to Lambert’s cosine law, since light arriving at steeper angles contributes less [9]. The result is an irradiance map that stores the average incoming light for each direction, which can be sampled during rendering (Fig. 2).

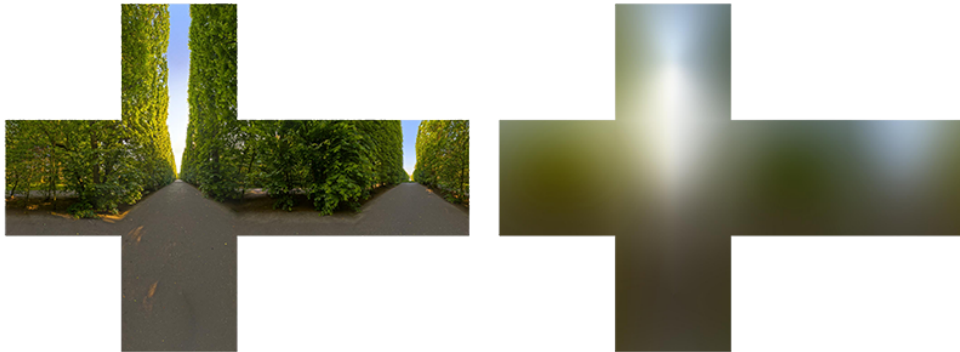


Figure 2: Environment radiance cubemap (left) and convolved irradiance cubemap (right) [9]

A *light probe* is a sample of the surrounding lighting at a specific point in a scene, stored as a texture such as a cubemap. Objects near the probe can use its data to approximate the ambient lighting at their location. In offline or pre-baked workflows, light probes are placed manually by artists or distributed on a grid, and their cubemaps are rendered once during a build step. Optimal placement is not trivial, as too few probes cause lighting discontinuities between objects while too many increase storage and computation cost [40]. The thesis uses a single probe that is regenerated each frame or at a configurable interval, which is sufficient for a stationary camera but would need further optimization to scale to dynamic multi-probe scenarios.

3.3 Techniques and Methods

Novel View Synthesis (NVS) is the task of generating images of a scene from camera view-points not present in the input data. Both NeRF [27] and 3DGS facilitate NVS by leveraging ML-based methods working with data generated by SfM. The thesis uses this capability to render the Gaussian environment from six directions for cubemap generation. NVS methods generally require 3D scene structure as input data. *Structure-from-Motion* is the process of estimating camera motion and scene structure from a sequence of images. A commonly used implementation in related work is COLMAP, a general-purpose SfM and Multi-View Stereo (MVS) pipeline [34]. SfM produces the initial camera poses and sparse point clouds that serve as input for training *3D Gaussian Splatting (3DGS)* [22], a recent method that uses modern machine learning to create optimized Gaussian Splatting scenes from realistic environment footage. Each Gaussian kernel stores position, covariance, opacity and color using SH coefficients. There is no formal method to determine if a 3DGS scene represents an environment or a discrete object, since the distinction is only based on convention and training input, where isolated objects can be produced by masking the background in the training images.

In contrast, *NeRF* represents a continuous radiance field defined implicitly via a Multi-Layer Perceptron (MLP) [27], whereas the method used by 3DGS produces a discretized radiance field. The continuous aspect of Neural Radiance Fields (NeRFs) helps optimization, but rendering requires a costly sampling method that can cause noisy output [22]. Because of this high computational cost, NeRFs are unsuitable for real-time rendering, which motivated the development of 3DGS. The thesis uses *Image-Based Lighting (IBL)*, a rendering technique that captures an omnidirectional representation of an environment scene as an image [10, 47].

This image is then sampled in a shader to represent directional ambient light. The thesis combines 3DGS rendering with IBL by generating an environment cubemap from the Gaussian scene and filtering it into an irradiance map.

The mesh objects in the thesis are shaded using Physically Based Rendering, which models surface appearance through the BRDF. The Cook-Torrance model [7] splits the BRDF into a diffuse term for scattered light and a specular term for view-dependent highlights. Burley [5] introduced the Disney BRDF, which parameterizes surface properties using artist-friendly values like roughness and metallic, and is widely adopted in real-time rendering and in related work on 3DGS relighting. For real-time IBL, Karis [21] proposed the split-sum approximation, which splits the lighting calculation into two precomputed parts so that the expensive integration does not have to be done per pixel at runtime. The thesis implementation uses the diffuse part of this pipeline by sampling an irradiance cubemap with the surface normal.

3.4 Technologies and Specifications

The *PLY* format stores graphical objects as a collection of vertices, faces and other elements with attached properties such as color and normal direction. The method proposed in 3DGS uses the *PLY* format to represent volumetric primitives, where the attached properties include covariance, opacity and view-dependent color in the form of Spherical Harmonics coefficients. To support degree-3 SH, 16 basis functions are stored for each color channel, resulting in 48 coefficients per Gaussian. The properties `f_dc_0`, `f_dc_1` and `f_dc_2` (direct color) correspond to the zero-frequency SH term, whereas the properties `f_rest_0` through `f_rest_44` make up all higher-order terms.

The thesis implementation is built on *Vulkan*, a cross-platform graphics and compute API [23] where new functionalities must be activated explicitly via physical device feature flags when creating a logical device. The Vulkan ecosystem provides several helper libraries used in the implementation: *Volk* is a meta-loader that allows dynamic loading of Vulkan entrypoints and simplifies the use of extensions [20], while the *Vulkan Memory Allocator (VMA)* provides higher-level functions for memory allocation and resource creation [1]. Both libraries are used in VVE and vkgs. The implementation makes use of *Dynamic Rendering*, which issues rendering commands without pre-created `VkRenderPass` or `VkFramebuffer` objects by specifying attachments at command recording time using `vkCmdBeginRenderingKHR`. This approach simplifies code and increases flexibility while still supporting deferred shading, where geometry buffers (G-buffers) are first written and then read for lighting computations. In addition to graphics pipelines, Vulkan provides *compute pipelines* that run general-purpose workloads on the GPU using compute shaders.

The Gaussian rendering pipeline uses compute shaders for most of its stages, including parsing, frustum culling, sorting and projection, before a final graphics pass draws the splats. The irradiance convolution filter is also implemented as a compute shader. The *Vienna Vulkan Engine* [16], an educational codebase showcasing different Vulkan rendering implementations, including Forward and Deferred Rendering, is extended by adding a Gaussian Splatting renderer and an irradiance cubemap lighting demonstration. The demo uses two rendering implementations: the Gaussian renderer for the environment and the Deferred renderer for a mesh-based object with PBR including IBL. The Gaussian renderer is based on *vkgs*, a Vulkan-based Gaussian Splatting viewer optimized for rendering speed [29], which implements two render modes, of which the thesis adapts the triangle list implementation. For alpha blending, vkgs uses *vulkan_radix_sort*, a header-only Vulkan radix sort library that implements a reduce-then-scan GPU algorithm [18]. The library requires Vulkan 1.4, so the VVE version requirement

is conditionally increased when building with Gaussian Splatting enabled, and the device features `storageBuffer16BitAccess` and `uniformAndStorageBuffer16BitAccess` are activated for SH coefficient storage.

4 Thesis Contribution

This thesis project adds a Gaussian Splatting renderer and an irradiance cubemap IBL generator implementation, and two demonstration applications written in Vulkan to the VVE codebase. The *vkgs* [29] codebase served as an inspiration and a starting point for recreating an optimized Vulkan renderer. The result is a hybrid rendering system combining 3DGS environments with deferred mesh rendering. Both renderers render to the same swapchain image using `VK_ATTACHMENT_LOAD_OP_LOAD` to preserve previous content. The runtime pipeline (Fig. 3) loads a trained 3DGS scene from a PLY file and renders it using the Gaussian renderer. To extract the environment cubemap, the same Gaussian rendering pipeline is re-run six times from the probe position, once for each cube face, using axis-aligned cameras with a 90-degree field of view. The resulting cubemap captures the radiance of the Gaussian environment as seen from the probe location. This cubemap is then filtered into an irradiance map, which the deferred renderer samples during its lighting pass to add the diffuse ambient contribution to the PBR result.

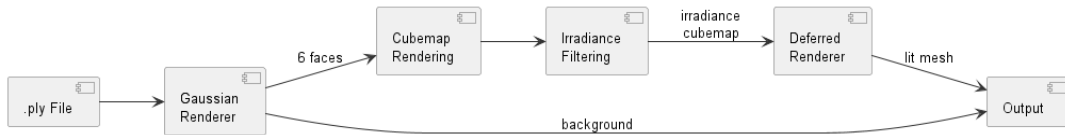


Figure 3: Overview of the hybrid rendering pipeline

4.1 Gaussian Splatting Renderer

Since optimized rendering of 3DGS scenes is not the focus and therefore out of scope of this thesis, the existing *vkgs* [29] implementation was used as a starting point. The *vkgs* codebase is a monolithic implementation that combines rendering, monitoring and Graphical User Interface (GUI) aspects into a single pipeline. Instead of cloning the dependency, relevant parts of the code were copied and adapted to fit the rendering pipeline of VVE. The adapted renderer is registered as a VVE System and communicates with the engine via message callbacks. The Gaussian rendering pipeline consists of multiple compute stages followed by a graphics pass: the PLY file is first parsed on the GPU to extract positions, covariance and SH coefficients, after which a rank pass performs frustum culling and generates depth keys for visible splats. The depth keys are then sorted back-to-front using `vulkan_radix_sort` [18] to ensure correct alpha blending, followed by an inverse index pass and a projection pass which computes screen-space positions and 2D covariance for each splat. The final graphics pass renders each splat as a screen-aligned quad using the triangle list mode, where four vertices per splat are generated in the vertex shader from a pre-computed index buffer. The OpenGL Shading Language (GLSL) shaders from *vkgs* were kept as-is and are compiled separately from the Slang shaders used by the VVE deferred renderer. Gaussian objects are represented as `GaussianSplat` components in the entity-component system, which store the PLY file path,

model transform and rendering parameters such as emissive intensity and splat count. This connects the renderer to VVE’s architecture. The entire Gaussian Splatting feature is conditionally compiled via the `VVE.GAUSSIAN_ENABLED` build flag, which ensures that the original VVE codebase remains unaltered when not building with Gaussian Splatting support (Fig. 4).



Figure 4: Application screenshot. Adapted Gaussian Splatting renderer integrated into VVE, rendering without hybrid lighting or mesh objects

4.2 Diffuse Irradiance Cubemap Generator

The thesis implementation provides two filter types for irradiance generation. The convolution filter performs cosine-weighted hemisphere sampling for each output texel using a Riemann sum approximation, which produces physically accurate diffuse irradiance at the cost of higher computation time. The implementation is based on the diffuse irradiance approach described by de Vries [9]. The convolved result of a sampled environment map is often called an *irradiance map*. The box filter relies on hardware texture sampling to downsample the environment cubemap, which produces visually similar results at a fraction of the computation cost (compare Fig. 5a and Fig. 5b). In order to generate the environment cubemap, the Gaussian Splatting scene is rendered six times from the probe location using 90-degree field-of-view cameras oriented along each cube face axis. The resulting irradiance cubemap is then passed to the deferred lighting shader, which samples it using the surface normal to add the diffuse ambient contribution to the PBR lighting result. Convolution is necessary for realistic ambient light behavior using environmental texture sampling; if the environment texture is not convolved, high-frequency signals in the texture could cause the lighting value associated with a specific normal vector to fluctuate, which would result in unexpected visual artifacts, e.g., inconsistent texture coloring on view-angle transforms.

4.3 Gaussian Splatting Cubemap Lighting Demo

The demonstration application renders a Gaussian Splatting environment scene with a traditional mesh-based object placed inside. A light probe is positioned at the object location to capture the surrounding illumination. The scene is configured via a `SceneConfig` structure,



Figure 5: Comparison of convolution filter (left) and box filter (right) on the Interior scene

which stores asset file paths, transform parameters, cubemap clear color and IBL update interval. At runtime, the demo retrieves the `RendererGaussian` system via `Engine::GetSystem()` and calls `GenerateCubemapIBL()` to regenerate the irradiance cubemap at the configured interval. The Gaussian environment transform can be adjusted at runtime using numpad controls for position and rotation, which allows visual inspection of how the ambient lighting on the mesh object responds to changes in the surrounding environment (Fig. 6a, Fig. 6b). FPS and GPU timing metrics are periodically logged to console output for performance monitoring.



Figure 6: Demonstration with the Bicycle scene using different filter types and mesh objects

4.4 Limitations

When generating the irradiance cubemap, a single stationary light probe is used and all objects share the same cubemap. Rotating the 3DGS environment was implemented for measuring the performance impact of real-time scene transformation and for debugging purposes, but translation was not implemented since spatially varying irradiance introduces additional complexity. Since the main focus of the thesis implementation was to demonstrate light interaction, the rendered images of the Gaussian and deferred rendering pipeline are composited by overwriting the swapchain image, resulting in the 3DGS environment always being rendered in the background. The adapted Gaussian Renderer is less efficient because of the usage of global synchronization patterns like `vkQueueWaitIdle` instead of fences or semaphores. The Fresnel-Schlick approximation is not applied when computing IBL. Configurations are applied at build time, i.e., changing rendering parameters or file paths requires rebuilding.

Furthermore, the lighting interaction between Gaussian Splatting environments and mesh-based objects is unidirectional, i.e., mesh objects receive ambient lighting from the Gaussian environment, but cannot influence it. Mesh objects do not cast shadows onto splats and are not captured in the environment cubemap, which means inter-reflections between meshes and the Gaussian environment are not modeled. The thesis implementation only computes diffuse irradiance, i.e., specular IBL in the form of pre-filtered environment maps is not implemented, which limits the accuracy of lighting on metallic or glossy surfaces. In order to reuse the graphics pipeline when creating cubemap resources, the image format is specified as `VK_FORMAT_B8G8R8A8_SRGB`. The usually preferred format for irradiance cubemaps is `VK_FORMAT_R16G16B16A16_SFLOAT`, which provides the precision needed for HDR rendering. Additionally, not all 3DGS scenes are fully enclosed environments. Scenes with incomplete coverage contain gaps where no Gaussian splats exist, which appear as the cubemap clear color in the captured environment map (Fig. 7). The implementation does not include inpainting or hole-filling, so these gaps directly affect the irradiance result.

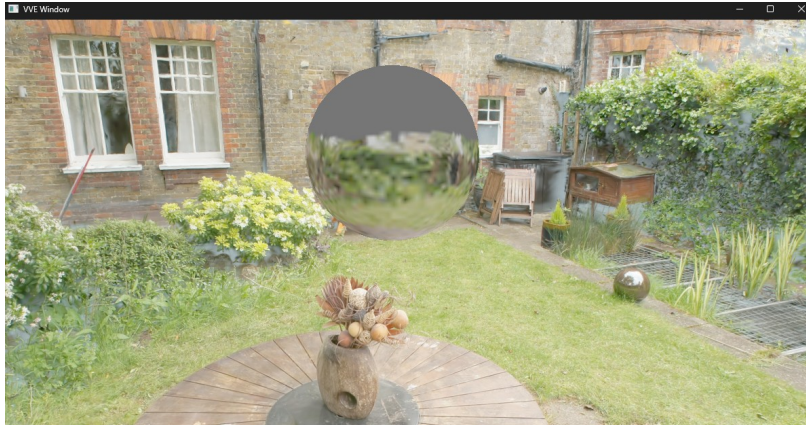


Figure 7: Garden scene with incomplete coverage: gaps in the 3DGS environment are visible as the clear color on the sphere’s irradiance

4.5 Usage of Generative AI

Generative Artificial Intelligence (AI) tools were used to assist research (e.g., explain technical terms or concepts, analyze grammar and phrasing, etc.), to analyze and debug code and application output, to document code sections in the thesis implementation, to extract hardcoded values into a configuration object and parametrize the demo application code, and to generate a Powershell script for automatic evaluation of different configuration setups. The thesis implementation logic and source code were authored manually. Code sections or files created or co-authored with the assistance of generative AI are annotated with comments and disclaimers.

5 Evaluation

The hybrid rendering pipeline was evaluated by benchmarking different combinations of Gaussian Splatting scenes, mesh objects, cubemap resolutions and irradiance filter types. Perfor-

mance was measured using frame rate and per-stage GPU timing to identify bottlenecks and scaling behavior. The results are discussed with respect to the research questions posed in the introduction.

5.1 Benchmark Setup

The hybrid rendering approach was evaluated through systematic benchmarking across multiple configurations. Technical evaluation combined quantitative performance metrics with visual inspection of the rendered output to verify correctness. The following parameters were varied to assess their impact on performance: *Irradiance cubemap resolution*, i.e., environment cubemap sizes of 256, 512, and 1024 pixels per face, with corresponding irradiance map resolutions of 16, 32, and 64 pixels, since higher resolutions capture finer environmental detail but increase computational cost. *Irradiance filter type*, i.e., convolution filter (cosine-weighted hemisphere sampling) versus box filter (hardware-accelerated downsampling), since the convolution filter produces physically accurate diffuse irradiance but requires significantly more computation. *IBL update interval*, i.e., regenerating the irradiance cubemap every frame (interval=1) versus every 10 frames (interval=10), since less frequent updates reduce GPU load at the cost of lighting responsiveness to environmental changes. *Gaussian splat count*, i.e., two scenes with different complexity, Interior Design (~50,000 splats) and Bicycle (~6 million splats), to evaluate scaling behavior. *Mesh complexity*, i.e., simple sphere (~1,000 triangles) versus Viking Room model (~100,000 triangles) to assess the impact of traditional geometry on overall performance.

Performance was measured using two complementary approaches. Frame rate (FPS) was computed as the average over one-second intervals to provide a measure of overall rendering throughput. For detailed computation time analysis, Vulkan’s GPU Timestamp Queries were used to measure individual compute and rendering stages. The measured stages include the vkgs rendering steps, i.e., rank (frustum culling and depth key generation), sort (radix sort for back-to-front ordering) and render (Gaussian splat rasterization), as well as the thesis implementation steps, i.e., cubemap (rendering the Gaussian environment to six cubemap faces) and irradiance (filtering the environment cubemap to produce diffuse irradiance). All metrics were logged to console output at one-second intervals during benchmark execution.

All benchmarks were conducted on a desktop system with the specifications listed in Table 1.

Table 1: Benchmark system specifications

Component	Specification
CPU	AMD Ryzen 9 3950X (16 cores)
GPU	NVIDIA RTX 3070 (8 GB)
Memory	64 GB RAM
Operating System	Windows 11 (64-bit)
Graphics API	Vulkan 1.4.335
GPU Driver	NVIDIA 591.74

The application was compiled in release mode using Microsoft Visual C++ (MSVC) 14.43.34808. Each benchmark configuration ran for 30 seconds with a fixed camera position to ensure consistent rendering load. Two Gaussian Splatting scenes were used: *Interior Design*, a compact indoor scene with approximately 50,000 splats, and *Bicycle*, a detailed outdoor scene from the MipNeRF-360 dataset [2] containing approximately 6 million splats.

5.2 Evaluation Results

Table 2 presents representative results across different scene configurations using the convolution filter. Gaussian splat count is the dominant factor affecting performance, as the Bicycle configuration (6M splats) results in 49 FPS at 512/32 resolution compared to 264 FPS for the Interior configuration at the same settings. At the highest resolution (1024/64), the Interior configuration is reduced to 77 FPS, since irradiance convolution cost (8.91 ms) becomes the bottleneck, while for the Bicycle configuration the frame rate is limited to 41 FPS as cubemap generation (17.80 ms) across six million splats dominates the frame time.

Table 2: Representative benchmark results across scene configurations (Convolution filter) with the same mesh object (Sphere)

Scene	Res.	Interval	FPS	Render [ms]	Cubemap [ms]	Irradiance [ms]
Interior	256/16	1	394.8	0.39	1.07	0.15
Interior	512/32	1	264.4	0.39	1.56	0.77
Interior	1024/64	1	76.6	0.40	3.58	8.91
Interior	1024/64	10	446.0	0.39	3.57	8.93
Bicycle	512/32	1	48.7	1.42	14.67	1.04
Bicycle	1024/64	1	40.6	1.42	17.80	2.37
Bicycle	1024/64	10	140.5	1.43	17.84	9.77

Resolution Scaling. Table 3 isolates the effect of IBL resolution on the Interior scene, comparing convolution and box filter approaches. The convolution filter scales approximately quadratically with irradiance resolution: increasing from 16 to 64 pixels per face raises irradiance computation from 0.15 ms to 8.91 ms (59× increase), since the hemisphere sampling must be performed for each output texel. In comparison to the convolution filter, the box filter maintains near-constant irradiance cost (0.01 to 0.02 ms) regardless of resolution, as it relies on hardware texture sampling rather than explicit integration, which results in 210 FPS at 1024/64 compared to 77 FPS with convolution.

Table 3: IBL resolution scaling on Interior scene (low splat count, Interval=1)

Cubemap	Irradiance	Filter	FPS	Cubemap [ms]	Irradiance [ms]
256	16	Conv	394.8	1.07	0.15
512	32	Conv	264.4	1.56	0.77
1024	64	Conv	76.6	3.58	8.91
256	16	Box	421.1	1.07	0.01
512	32	Box	337.5	1.56	0.01
1024	64	Box	209.5	3.24	0.02

Splat Count Impact. Table 4 compares the two test scenes at identical IBL settings to quantify the impact of Gaussian splat count. The 120× higher splat count of the Bicycle configuration (6M vs 50k) results in proportionally increased costs across the Gaussian Splatting rendering pipeline: rank time increases from 0.01 ms to 0.20 ms, sort time from 0.08 ms to 0.44 ms, and render time from 0.40 ms to 1.42 ms. Cubemap generation increases from 3.58 ms to 17.80 ms, as each of the six cubemap faces must render all visible splats. The irradiance

convolution time, however, *decreases* for the Bicycle configuration (2.37 ms vs 8.91 ms) despite identical output resolution. Since the convolution performs the same work in both cases, this may be caused by GPU frequency scaling triggered by the heavier cubemap workload.

Table 4: Gaussian splat count impact on IBL generation (1024/64, Conv, Interval=1)

Scene	Splats	FPS	Rank [ms]	Sort [ms]	Render [ms]	Cubemap [ms]	Irr. [ms]
Interior	~50k	76.6	0.01	0.08	0.40	3.58	8.91
Bicycle	~6M	40.6	0.20	0.44	1.42	17.80	2.37

Update Interval Impact. Table 5 shows the performance impact of skipping IBL regeneration for some frames. Reducing the update frequency from every frame to every 10 frames results in speedups of 3.5 to 5.9 \times depending on scene complexity. The Interior configuration benefits most (5.9 \times), since IBL generation dominates its frame time at high resolution. The Bicycle configuration shows smaller improvement (3.6 \times), as Gaussian Splatting rendering itself constitutes a larger fraction of frame time.

Table 5: IBL update interval impact on frame rate (1024/64 resolution, Convolution)

Scene	Mesh	Interval=1	Interval=10	Speedup
Interior	Sphere	76.6	446.0	5.8 \times
Interior	Viking	76.2	447.9	5.9 \times
Bicycle	Sphere	40.6	140.5	3.5 \times
Bicycle	Viking	40.5	144.6	3.6 \times

Mesh Complexity Impact. Table 6 compares the effect of traditional mesh complexity on hybrid rendering performance. Replacing the simple sphere (1k triangles) with the Viking Room model (100k triangles) produces negligible performance differences, with less than 1% variation for both scene configurations. This indicates that IBL generation and Gaussian Splatting rendering dominate frame time, while mesh rasterization, even for moderately complex geometry, remains a minor contributor.

Table 6: Mesh complexity impact on performance (1024/64, Conv, Interval=1)

Scene	Mesh	Triangles	FPS	Difference
Interior	Sphere (simple)	~1k	76.6	–
Interior	Viking Room	~100k	76.2	–0.5%
Bicycle	Sphere (simple)	~1k	40.6	–
Bicycle	Viking Room	~100k	40.5	–0.2%

5.3 Discussion

The benchmark results reveal that the cost of the hybrid pipeline is not distributed evenly across its stages. Rendering the Gaussian environment to six cubemap faces is the most expensive step when the scene contains many splats, while the irradiance filter becomes the

bottleneck only for small scenes at high output resolutions. This means that in practice, the choice of filter type matters most for lightweight environments, where convolution can consume more time than the cubemap pass itself. For dense scenes like Bicycle, the filter choice has little effect because the cubemap rendering already limits the frame rate.

The box filter produces visually comparable ambient lighting at a fraction of the convolution cost. Since diffuse irradiance is inherently low-frequency, the loss of accuracy from skipping cosine-weighted sampling is difficult to perceive. This suggests that for real-time applications where visual plausibility is sufficient, the box filter is the more practical choice, while convolution remains useful when physically accurate irradiance is required.

Reducing the update interval is the simplest way to recover performance. Because ambient lighting from a static environment does not change between frames, regenerating the irradiance cubemap every frame is unnecessary in most scenarios. Updating every 10 frames recovers most of the frame budget with no visible difference in static scenes. An adaptive strategy that triggers regeneration only when the camera or environment changes could further reduce wasted computation. Mesh complexity has negligible impact on frame time because the Gaussian Splatting stages and cubemap generation dominate the frame budget, so the pipeline could support more complex mesh geometry without measurable performance loss. The irradiance convolution time is lower for the Bicycle configuration despite identical output resolution, likely due to GPU frequency scaling from the heavier cubemap workload, which introduces minor measurement variance but does not affect the overall conclusions. Additional findings are presented by answering the thesis research questions given the insights gained from the benchmark results.

(A1) Real-time lighting feasibility: Yes, real-time illumination is achieved using standard 3DGS training output without modification. Unmodified PLY files are loaded directly; no retraining or compression needed. Although Spherical Harmonics are in principle capable of capturing diffuse lighting, the SH produced by 3DGS represent the learned irradiance contribution for one specific splat, not the environment as a whole. SH also inherently suffer from visual artifacts caused by high-frequency components. Using a light probe to generate an environment cubemap provides an uncompressed scene view that can be analyzed, processed and filtered. The thesis uses separate rendering passes for Gaussian Splatting and deferred mesh rendering, composited onto the same swapchain image via cubemap-based IBL, maintaining a unidirectional data flow from the Gaussian Splatting renderer to the deferred renderer.

(A2) Performance impact and integration challenges: Gaussian splat count is the primary factor affecting IBL generation performance, since cubemap generation requires rendering the full Gaussian environment for each of the six cubemap faces. Irradiance convolution scales approximately quadratically with output resolution, which makes it the dominant cost factor at high resolutions for low splat-count scenes. The box filter provides a potential alternative with near-constant computation time. Skipping IBL regeneration for some frames provides up to $6\times$ speedup, while mesh complexity has negligible impact on overall frame time. The integration of the vkgs rendering pipeline into VVE required conditionally increasing the Vulkan version requirement from 1.3 to 1.4 and activating additional device features for SH coefficient storage, all guarded by the `VVE_GAUSSIAN_ENABLED` build flag. The monolithic vkgs codebase was decomposed into the VVE System architecture using message callbacks, and its traditional `VkRenderPass` objects were adapted to the Dynamic Rendering approach used by VVE.

6 Conclusion

This thesis demonstrated that real-time diffuse IBL from 3DGS environments is feasible using cubemap-based light probing without retraining or modifying the Gaussian scene data. The implementation renders a Gaussian Splatting environment and a mesh-based object together in a single Vulkan pipeline, where the mesh receives ambient lighting extracted from the surrounding environment via an irradiance cubemap. The evaluation confirmed that Gaussian splat count is the dominant performance factor, while mesh complexity has negligible impact. The box filter provides a practical alternative to convolution with near-constant computation cost, and reducing the IBL update interval offers significant speedups for scenes with static or slowly changing lighting. Unlike most related work which relies on ML-based inverse rendering to decompose materials and lighting from 3DGS scenes, this approach uses only traditional rendering techniques applied at runtime. The method is limited to diffuse lighting from a single probe location, but serves as a baseline that can be extended in several directions.

Several enhancements could build on this work. Multiple light probes could be placed at different spatial positions inside the environment to capture spatial variations in illumination, which would improve lighting accuracy for larger scenes. Specular IBL in the form of pre-filtered environment maps could be added to support glossy and metallic surface reflections. Depth compositing between the Gaussian Splatting environment and mesh-based objects could be investigated to allow mutual occlusion. An adaptive IBL update strategy based on camera movement or environment changes could reduce unnecessary recomputation for static scenes. The cubemap format could be changed to `VK_FORMAT_R16G16B16A16_SFLOAT` to enable HDR rendering. Shadow mapping and ambient occlusion between mesh objects and Gaussian Splatting environments could further improve realism, potentially by rendering a transparent mesh-based copy of the 3DGS scene (e.g., converted to mesh by methods proposed in related work [14, 41, 44]) that is exclusively used for lighting interaction. Given a set of discrete light probes at fixed positions, an interpolation function could be learned using ML methods to approximate lighting at intermediate positions, similar to the learnable probe approaches proposed by Teuber et al. [40] and Di Sario et al. [33].

References

- [1] AMD. Vulkan memory allocator. <https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator>, 2017.
- [2] BARRON, J. T., MILDENHALL, B., VERBIN, D., SRINIVASAN, P. P., AND HEDMAN, P. Mip-nerf 360: Unbounded anti-aliased neural radiance fields, 2022.
- [3] BLINN, J. F., AND NEWELL, M. E. Texture and reflection in computer generated images. *Commun. ACM* 19, 10 (Oct. 1976), 542–547.
- [4] BOLDUC, C., HOLD-GEOFFROY, Y., SHU, Z., AND LALONDE, J.-F. Gaslight: Gaussian splats for spatially-varying lighting in hdr, 2025.
- [5] BURLEY, B. Physically-based shading at disney.
- [6] CHEN, H., LIN, Z., AND ZHANG, J. Gi-gs: Global illumination decomposition on gaussian splatting for inverse rendering, 2025.
- [7] COOK, R. L., AND TORRANCE, K. E. A reflectance model for computer graphics. *ACM Trans. Graph.* 1, 1 (Jan. 1982), 7–24.
- [8] DAI, P., XU, J., XIE, W., LIU, X., WANG, H., AND XU, W. High-quality surface reconstruction using gaussian surfels, 2024.
- [9] DE VRIES, J. Diffuse irradiance — ibl / pbr. <https://learnopengl.com/PBR/IBL/Diffuse-irradiance>, 2026. Accessed: 2026-01-23.
- [10] DEBEVEC, P. Image-based lighting. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, Association for Computing Machinery, p. 3–es.
- [11] GAO, J., GU, C., LIN, Y., LI, Z., ZHU, H., CAO, X., ZHANG, L., AND YAO, Y. Relightable 3d gaussians: Realistic point cloud relighting with brdf decomposition and ray tracing, 2024.
- [12] GIBBS, J. W. Fourier’s series. *Nature* 59, 606–606.
- [13] GREENE, N. Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.* 6, 11 (Nov. 1986), 21–29.
- [14] GUÉDON, A., AND LEPETIT, V. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering, 2023.
- [15] GUÉDON, A., AND LEPETIT, V. Gaussian frosting: Editable complex radiance fields with real-time rendering, 2024.
- [16] HLAVACS, H. The vienna vulkan engine (vve): A vulkan based render engine. <https://github.com/hlavacs/ViennaVulkanEngine>, 2019. Accessed: 2026-02-16.
- [17] HUANG, B., YU, Z., CHEN, A., GEIGER, A., AND GAO, S. 2d gaussian splatting for geometrically accurate radiance fields, 2025.
- [18] JAESUNG. Vulkan radix sort, 2023. Accessed: 2026-01-22.

- [19] JIN, H., LIU, I., XU, P., ZHANG, X., HAN, S., BI, S., ZHOU, X., XU, Z., AND SU, H. Tensor: Tensorial inverse rendering, 2024.
- [20] KAPOULKINE, A. volk: Meta loader for Vulkan API. GitHub repository, 2025. MIT License.
- [21] KARIS, B. Real shading in unreal engine 4 by.
- [22] KERBL, B., KOPANAS, G., LEIMKÜHLER, T., AND DRETTAKIS, G. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics* 42, 4 (July 2023).
- [23] KHRONOS VULKAN WORKING GROUP. Vulkan 1.4.335 specification. <https://registry.khronos.org>, 2025. Accessed: 2026-01-22.
- [24] LI, J., WU, Z., ZAMFIR, E., AND TIMOFTE, R. Recap: Better gaussian relighting with cross-environment captures, 2025.
- [25] LIANG, Z., ZHANG, Q., FENG, Y., SHAN, Y., AND JIA, K. Gs-ir: 3d gaussian splatting for inverse rendering, 2024.
- [26] LIU, Z., OUYANG, H., WANG, Q., CHENG, K. L., XIAO, J., ZHU, K., XUE, N., LIU, Y., SHEN, Y., AND CAO, Y. Infusion: Inpainting 3d gaussians via learning depth completion from diffusion prior, 2024.
- [27] MILDENHALL, B., SRINIVASAN, P. P., TANCİK, M., BARRON, J. T., RAMAMOORTHI, R., AND NG, R. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.
- [28] PAN, L., BARÁTH, D., POLLEFEYS, M., AND SCHÖNBERGER, J. L. Global structure-from-motion revisited, 2024.
- [29] PARK, J. vkgs: Vulkan-based gaussian splatting viewer. <https://github.com/jaesung-cs/vkgs>, 2026. Accessed: 2026-01-01.
- [30] PORTER, T., AND DUFF, T. Compositing digital images. *SIGGRAPH Comput. Graph.* 18, 3 (Jan. 1984), 253–259.
- [31] RAMAMOORTHI, R., AND HANRAHAN, P. An efficient representation for irradiance environment maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, Association for Computing Machinery, p. 497–500.
- [32] SAITO, T., AND TAKAHASHI, T. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 197–206.
- [33] SARIO, F. D., REBAIN, D., VERBIN, D., GRANGETTO, M., AND TAGLIASACCHI, A. Spherical voronoi: Directional appearance as a differentiable partition of the sphere, 2025.
- [34] SCHÖNBERGER, J. L., AND FRAHM, J.-M. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

- [35] SCHÖNBERGER, J. L., ZHENG, E., POLLEFEYS, M., AND FRAHM, J.-M. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)* (2016).
- [36] SHI, Y., WU, Y., WU, C., LIU, X., ZHAO, C., FENG, H., ZHANG, J., ZHOU, B., DING, E., AND WANG, J. Gir: 3d gaussian inverse rendering for relightable scene factorization, 2024.
- [37] SKOROKHOV, V., DURASOV, N., AND FUA, P. D3dr: Lighting-aware object insertion in gaussian splatting, 2025.
- [38] SONG, Y., LIN, H., LEI, J., LIU, L., AND DANILIDIS, K. Hdgs: Textured 2d gaussian splatting for enhanced scene rendering, 2024.
- [39] SRINIVASAN, P. P., DENG, B., ZHANG, X., TANCIK, M., MILDENHALL, B., AND BARRON, J. T. Nerv: Neural reflectance and visibility fields for relighting and view synthesis, 2020.
- [40] TEUBER, M., KUNERT, C., SCHWANDT, T., AND BROLL, W. Geometry-based light probe placement for realtime lighting in gaussian splatting environments: Geometry-based light probe placement for realtime lighting in gaussian splatting environments. *Vis. Comput.* 41, 14 (Sept. 2025), 11783–11795.
- [41] TOBIASZ, R., WILCZYŃSKI, G., MAZUR, M., TADEJA, S., AND SPUREK, P. Mesh-splats: Mesh-based rendering with gaussian splatting initialization, 2025.
- [42] TURK, G. The ply polygon file format. Tech. rep., UNC Computer Science Technical Report, 1994. Accessed: 2026-01-xx.
- [43] TURKULAINEN, M., REN, X., MELEKHOV, I., SEISKARI, O., RAHTU, E., AND KANALALA, J. Dn-splatter: Depth and normal priors for gaussian splatting and meshing, 2024.
- [44] WACZYŃSKA, J., BORYCKI, P., TADEJA, S., TABOR, J., AND SPUREK, P. Games: Mesh-based adapting and modification of gaussian splatting, 2024.
- [45] WANG, Y., CHEN, S., AND YI, R. Sg-splatting: Accelerating 3d gaussian splatting with spherical gaussians, 2024.
- [46] WESTOVER, L. A. *Splatting: A parallel, feed-forward volume rendering algorithm*. PhD thesis, Chapel Hill, NC, USA, 1991.
- [47] WIKIPEDIA. Image-based lighting — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Image-based%20lighting&oldid=1276444736>, 2025. [Online; accessed 2026-01-22].
- [48] WU, T., SUN, J.-M., LAI, Y.-K., MA, Y., KOBELT, L., AND GAO, L. Deferreddgs: Decoupled and editable gaussian splatting with deferred shading, 2024.
- [49] XU, T., REN, X., AND WU, E. The power of box filters: Real-time approximation to large convolution kernel by box-filtered image pyramid. In *SIGGRAPH Asia 2019 Technical Briefs* (New York, NY, USA, 2019), SA '19, Association for Computing Machinery, p. 1–4.

- [50] YAO, Y., ZENG, Z., GU, C., ZHU, X., AND ZHANG, L. Reflective gaussian splatting, 2025.
- [51] YE, K., HOU, Q., AND ZHOU, K. 3d gaussian splatting with deferred reflection. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers 24* (July 2024), SIGGRAPH '24, ACM, p. 1–10.
- [52] ZHONG, H., WANG, C., ZHANG, J., AND LIAO, J. Generative object insertion in gaussian splatting with a multi-view diffusion model. *Visual Informatics 9*, 2 (June 2025), 100238.

Acronyms

- 3DGS** 3D Gaussian Splatting. 3–12, 14, 15, 19, 20
- 3D CG** 3D Computer Graphics. 3–5, 8
- AI** Artificial Intelligence. 15
- BRDF** Bidirectional Reflectance Distribution Function. 7, 8, 11
- BVH** Bounding Volume Hierarchy. 6
- FPS** Frames per Second. 4, 5, 14, 16–18
- GLSL** OpenGL Shading Language. 12
- GPU** Graphics Processing Unit. 11, 12, 14, 16, 18, 19
- GUI** Graphical User Interface. 12
- HDR** High Dynamic Range. 5–7, 15, 20
- IBL** Image-Based Lighting. 3–7, 9–12, 14–20
- ML** Machine Learning. 4, 10, 20
- MLP** Multi-Layer Perceptron. 10
- MSVC** Microsoft Visual C++. 16
- MVS** Multi-View Stereo. 10
- NeRF** Neural Radiance Field. 4, 6, 10
- NVS** Novel View Synthesis. 10
- PBR** Physically Based Rendering. 3, 4, 11–13
- PLY** Polygon File Format. 5, 11, 12, 19
- SDF** Signed Distance Field. 7
- SfM** Structure-from-Motion. 5, 10
- SH** Spherical Harmonics. 3, 5–12, 19
- SV** Spherical Voronoi. 7
- VMA** Vulkan Memory Allocator. 11
- VR** Virtual Reality. 3
- VVE** Vienna Vulkan Engine. 3, 4, 11–13, 19

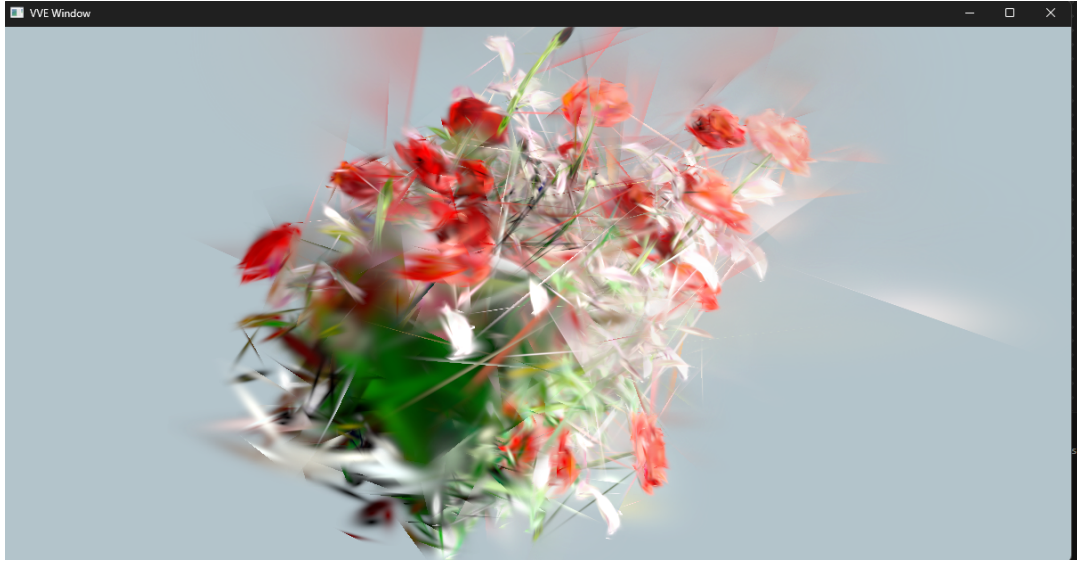


Figure 8: Bonus image. During the development process of the thesis project, the author used the wrong `VkPrimitiveTopology` when implementing the Gaussian Splatting renderer. The screenshot shows the result of using `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP` instead of `VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST`.