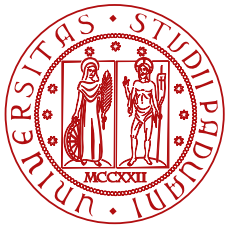


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITY OF PADOVA
INFORMATION ENGINEERING DEPARTMENT

MASTER DEGREE IN ICT FOR INTERNET AND MULTIMEDIA

Real-time XR visualization of 3D Gaussian Splatting models

AUTHOR

Chiara Schiavo

Student ID 2097337

SUPERVISOR

Prof. Simone Milani

University of Padova

CO-SUPERVISOR

Dr. Elena Camuffo

University of Padova

ACADEMIC YEAR 2023/2024
DATE 23/10/2024

Ai miei genitori
che mi hanno insegnato l'amore
Ad Alessio
con cui lo vivo ogni giorno

Abstract

The demand for high-quality, immersive experiences in eXtended Reality (XR) environments has increased significantly, driven by applications in entertainment, education, healthcare, robotics and industrial design. XR technologies rely heavily on real-time rendering of 3D scenes to create interactive and responsive experiences. However, achieving both visual fidelity and computational efficiency remains a major challenge, particularly on mobile and wearable devices with limited resources. Emerging approaches, such as 3D Gaussian Splatting, are transforming how 3D scenes are visualized in real time. These innovations offer the potential to enhance user experiences and expand the practical applications of XR technologies across various domains.

This thesis explores the application of 3D Gaussian Splatting in real-time XR visualization, proposing an adaptive Level of Detail system to optimize rendering performance. By dynamically adjusting scene complexity based on user position and the importance of objects within the scene, this approach seeks to enhance both visual quality and computational efficiency. Semantic segmentation is leveraged to prioritize and allocate resources effectively, ensuring that detailed rendering is focused on key areas while less important regions are simplified. The integration of these techniques allows for high-quality visualizations that adapt to real-time requirements, enabling for XR applications that balance computational constraints with the need for high visual fidelity.

Contents

List of Acronyms

3DGS 3D Gaussian Splatting

AR Augmented Reality

BRDF Bidirectional Reflectance Distribution Function

CAD Computer-Aided Design

CGI Computer-Generated Imagery

CNN Convolutional Neural Network

CPU Computer Processing Unit

DL Deep Learning

FPS Frame Per Second

GPU Graphic Processing Unit

LOD Level Of Detail

ML Machine Learning

MLP Multi-Layer Perceptron

MR Mixed Reality

MSV Multi-View Stereo

NeRF Neural Radiance Field

NVS Novel View Synthesis

PBR Physically Based Rendering

PSNR Peak Signal-to-Noise Ratio

SfM Structure from Motion

SSIM Structural Similarity Index Measure

VR Virtual Reality

XR eXtended Reality

1

Introduction

In recent years, the growth of eXtended Reality (XR) applications has transformed how users interact with digital content. Thanks to the widespread availability of mobile and wearable devices, these technologies have become increasingly accessible, opening up new use cases in fields such as entertainment, education, design, and healthcare. However, creating high-quality immersive experiences requires real-time processing of complex 3D models that must dynamically adapt to the user's position and movements. This presents significant challenges in terms of computational efficiency and visual quality, especially on resource-constrained devices like headsets and smartphones.

One of the main challenges in real-time visualization for XR environments is balancing visual fidelity with the constraints of available computational resources. The perceived image quality by the user depends not only on the ability to render detailed 3D models but also on the precise synchronization between the user's movements and the displayed content. Even a slight misalignment between the user's movements and the scene visualization can negatively affect the experience. This requires advanced rendering techniques that maintain high performance without compromising visual quality.

Traditionally, 3D model rendering for XR applications relied on explicit representations like point clouds, meshes, and voxels, which require significant computational power for real-time rendering. However, the evolution of neural-based rendering techniques has led to more efficient and flexible solutions. In this context, approaches such as Neural Radiance Fields (NeRF) and 3D Gaussian Splatting (3DGS) are emerging as promising solutions for representing and visualizing complex 3D scenes. The NeRF method, introduced in 2020, allows the synthesis of 3D scene views from a set of 2D images using a radiance field learned by a neural network. While NeRF has achieved remarkable results in terms of visual quality, its implementation is particularly computationally demanding, making it difficult to apply in XR scenarios where real-time responsiveness is essential. On the other hand, the 3D Gaussian Splatting method, introduced in 2023, represents a

significant advancement in real-time 3D scene visualization. This approach uses a set of 3D Gaussians distributed in space to represent the scene's geometry and appearance. The flexibility of Gaussians allows for continuous and differentiable scene representations, maintaining a high level of visual detail while reducing computational load compared to traditional or neural-based methods like NeRF. This makes 3DGS particularly suitable for XR applications on mobile and wearable devices, where hardware resources are limited.

1.1 Proposed Contribution

This thesis focuses on the application of 3D Gaussian Splatting for real-time visualization in XR environments. The main objective is the development of a semantic-based dynamic rendering framework that optimizes visual quality while addressing the challenges posed by limited computational resources and bandwidth constraints typical of mobile and wearable XR devices. By dynamically controlling the level of detail based on the user's position and the complexity of semantic classes recognized in the scene, the system presented in this thesis aims to enhance the user experience in XR scenarios by reducing the computational burden without sacrificing visual quality.

The thesis is organized as follows. Chapter 2 provides an overview of the theoretical background on rendering techniques, with a focus on real-time 3D scene rendering in XR applications, highlighting the key challenges and limitations. Chapter 3 discusses state-of-the-art rendering methods, specifically focusing on NeRF and 3DGS. In Chapter 4, we analyze the datasets and metrics used for evaluating rendering performance and visual quality. Chapter 5 presents the proposed method, describing the implementation of the adaptive LOD system based on 3DGS. Experimental results and performance evaluations are discussed in Chapter 6. Finally, conclusions and future research directions are provided in Chapter 7.

2

Rendering

Rendering is the computational process of generating a 2D image from a 3D model or scene description, and plays an important role in computer graphics. It involves the simulation of light interactions with objects, materials, and surfaces to produce a visual output. The rendering pipeline typically encompasses stages such as geometric transformations, lighting calculations, texture mapping, and shading, leveraging both central processing units (CPUs) and graphics processing units (GPUs) for efficiency.

In computer graphics, rendering can be classified into two main categories: offline rendering and real-time rendering. Offline rendering, typically used in applications like film production, architectural visualization, and high-quality simulations, prioritizes visual fidelity over speed. This method employs computationally intensive techniques such as ray tracing, path tracing, and global illumination, allowing for physically accurate lighting, shadows, and reflections. Offline rendering can take hours or even days to generate a single frame, as seen in CGI effects for movies, where realism is paramount. Real-time rendering is essential for interactive applications like video games, virtual reality, and simulations, where maintaining high frame rates is fundamental to ensure smooth user interaction. To achieve this, real-time rendering relies on optimized algorithms like rasterization, approximate lighting models, and hardware acceleration using GPUs, sacrificing some visual precision for performance.

2.1 Graphic Pipeline

In the context of computer graphics, the visualization of a virtual environment starts from a three-dimensional model of the scene to generate one or more views to be displayed on a two-dimensional image on a screen. The scene can be created from real-world data, such as photographs or sensor inputs, in this case, the process is known as 3D reconstruction, and focus on the goal of obtaining a model that is as faithful and realistic as possible. Alternatively, the model can be generated digitally using modeling software, without necessarily drawing inspiration from real-world elements.

In the following, we will primarily focus on 3D reconstruction, as it represents the starting point for acquiring virtual models from real data. However, the discussion of rendering remains applicable to models obtained through 3D modeling, as the principles and techniques employed in the rendering pipeline are relevant to both modes of view synthesis.

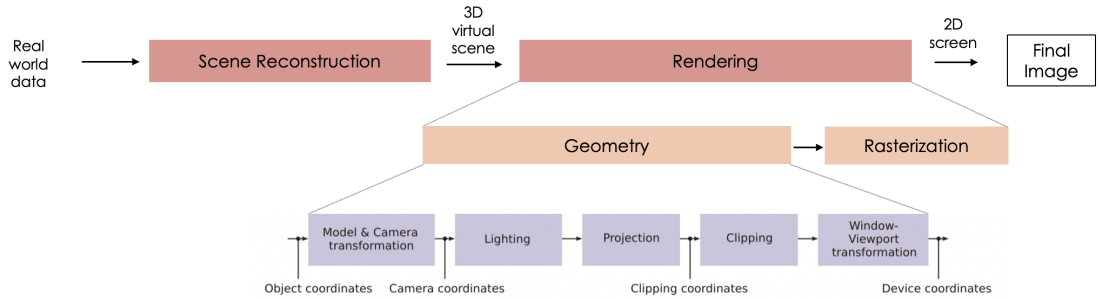


Figure 2.1: Overview of the graphic pipeline. Real-world data is first processed through Scene Reconstruction to create a 3D virtual scene. This scene is then passed to the Rendering stage, which involves Geometry transformations and Rasterization. Key steps in this process include model and camera transformations, lighting, projection, clipping, and viewport transformations, ultimately producing the final rendered image.

2.1.1 3D Reconstruction

3D reconstruction is the process of automatically or semi-automatically generating a model from real-world data, with maximum fidelity and realism. Typically, 3D reconstruction is achieved by traditional algorithms such as photogrammetry, Multi View Stereo (MVS) and Structure from Motion (SfM) algorithms. In these cases, it involves extracting the geometric and appearance information from a series of 2D images, typically captured from different viewpoints. This information is then used to create a 3D model of the scene. Specifically, Structure from Motion reconstructs the 3D structure of a scene from a set of overlapping 2D images as input. First, it identifies common features across these images, such as corners, edges, or specific patterns; then it estimates the position and orientation (pose) of the camera for each image based on the identified features and

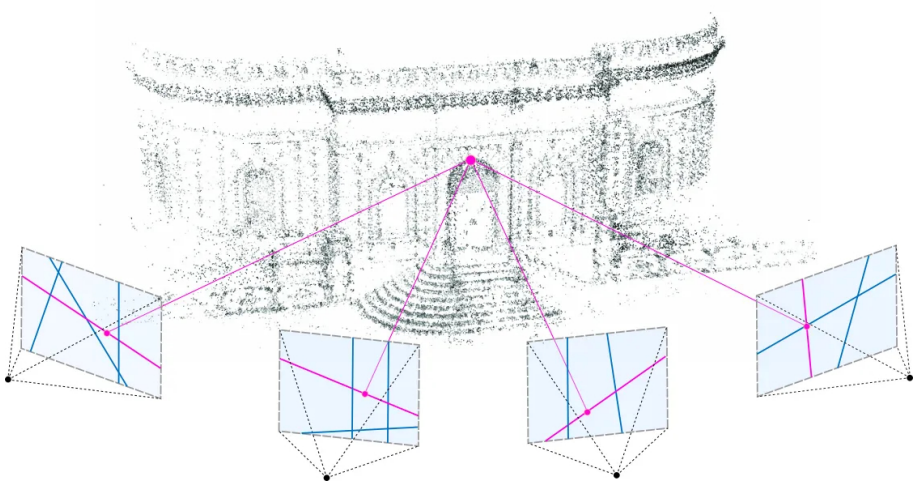


Figure 2.2: 3D scene reconstruction with Structure from Motion [16]

how they appear from different viewpoints. By having corresponding features in multiple images and the camera poses, it performs triangulation to determine the 3D location of these features in the scene. Lastly, the SfM algorithms use the 3D positioning of these features to build a 3D model of the scene which can be a point cloud representation or a more detailed mesh model.

2.1.1.1 Scene representation

Scene representation encompasses various methodologies for encoding spatial and visual characteristics of three-dimensional environments.

Explicit scene representation Traditionally, 3D shapes are described using explicit representations such as depth images, voxel grids, point clouds, or meshes.

(a) Point cloud **(b)** Voxel **(c)** Mesh

Figure 2.3: The Stanford Bunny model in three different explicit representation

Depth images capture the distance between the camera and the object at each pixel, encoding 3D geometry from a specific viewpoint. Layered depth images expand on this by incorporating multiple depth maps with color information to depict a scene.

Point clouds (Figure 2.3a) consist of a collection of vertices in 3D space, defined by a triplet of coordinates (x, y, z). These points collectively describe the geometry of an object

or an entire scene and are commonly generated by 3D scanners, such as LiDAR or RGB-D sensors, from multiple viewpoints.

Volumetric models (Figure 2.3b) are regular data structures that capture the three-dimensional volume of an object or scene. They serve as an intuitive extension of 2D images into the third dimension, where individual elements, called voxels, are analogous to pixels in 2D. Just as a pixel is a subdivision of a 2D space, a voxel represents a discrete unit within a 3D space. Voxel grids are a common implementation of this concept, where a regular 3D grid represents the object or scene. These grids can be constructed from point clouds through voxelization, which groups nearby points into individual voxels.

Boundary representation (B-rep) defines the shape of an object by specifying its boundaries through vertices, edges, and faces. The most common B-rep model is the 3D mesh (Figure 2.3c), a geometric structure extensively used in computer graphics. A 3D mesh encodes the surface geometry of objects through polygonal faces, typically triangular or quadrilateral in shape. When a mesh consists of a single type of polygon, such as triangles, it is referred to as a regular mesh. Unlike point clouds, which only provide vertex locations, meshes also describe the surface geometry, offering more detailed information about an object's structure.

Implicit scene representation Implicit scene representations do not store geometry directly; instead, they use continuous functions to describe it. Since implicit functions are often not analytically tractable, they are typically approximated by neural networks. In this context, implicit neural representations, such as neural fields, offer a novel way to parameterize signals across a wide range of domains.

Radiance Field A radiance field is a representation of light distribution in a three-dimensional space, which captures how light interacts with surfaces and materials in the environment. Mathematically, a radiance field can be described as a function $L : \mathbb{R}^5 \rightarrow \mathbb{R}^+$ where $L(x, y, z, \theta, \varphi)$ maps a point in space (x, y, z), and a direction specified by spherical coordinates (θ, φ), to a non-negative radiance value. Radiance fields can be represented in both forms: implicit and explicit.

In an implicit radiance field, the light distribution in a scene is not stored directly but instead learned through neural networks. The neural network maps spatial coordinates (x, y, z) and viewing directions (θ, φ) to color and density values. The radiance at any point in space is computed on-the-fly by querying the network. This approach enables continuous and differentiable scene representations, but requires higher computational power.

Conversely, in an explicit radiance field, the light distribution is stored directly in a predefined data structure, such as voxel grids or point clouds. Each element within this structure stores radiance information for its respective location in space. While this

method allows for faster access to radiance data, it typically demands more memory and may offer lower resolution compared to implicit representations. A radiance function in this context can be written as

$$L_{\text{explicit}}(x, y, z, \theta, \varphi) = \text{DataStructure}[(x, y, z)] \cdot f(\theta, \varphi)$$

where the data structure might represent a voxel grid or point cloud.

2.1.1.2 Camera model

In the context of 3D reconstruction, accurate camera models and their calibration techniques are essential for capturing and interpreting the spatial relationship between the real world and its digital representation. Camera models describe how 3D points in the environment $(X_W, Y_W, Z_W, 1)$ are projected onto a 2D image plane (u, v, w), enabling the reconstruction of the scene's geometry. The parameters associated with camera models can be broadly categorized into intrinsic and extrinsic parameters, both of which are essential for accurately capturing and reconstructing three-dimensional scenes.

Intrinsic parameters define the internal characteristics of a camera, including focal length $(f_x,\ f_y)$, sensor size, and lens distortion. These parameters determine how a camera captures light and projects it onto its sensor, influencing the overall quality and accuracy of the images produced. For instance, the focal length affects the field of view, while lens distortion can lead to image warping. Accurately calibrating these intrinsic parameters ensures that the depth information derived from images reflects the true geometry of the scene.

Extrinsic parameters describe the position $(t_{3\times1})$ and orientation $(R_{3\times3})$ of the camera in the three-dimensional space relative to the scene being captured. These parameters are critical for establishing the relationship between multiple images taken from different viewpoints, as they determine how the camera's coordinate system aligns with the world coordinate system. Accurate extrinsic parameters allow for the correct alignment of images and depth data, enabling the generation of coherent 3D models.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} \begin{pmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \tag{2.1}$$

2.1.2 Rendering pipeline

The purpose of the rendering pipeline is to systematize the rendering process, ensuring that all aspects of a scene, from geometry to lighting, are processed in a logical order.

This modular approach allows for optimization at each stage, ensuring that even complex scenes can be rendered. In traditional offline rendering, where quality matters more than speed, the pipeline enables effects like ray tracing for realistic shadows and reflections. Real-time rendering pipelines are optimized for speed, using techniques like z-buffering, back-face culling, and Level Of Detail (LOD) to reduce the computational load without sacrificing too much visual quality. Though the specific steps may vary depending on the renderer or game engine being used, the core stages remain similar.

2.1.2.1 Geometry Processing

Initially, objects are defined in model space, which locates them relative to their own coordinate systems. To prepare the scene for display on a 2D screen, the first transformations occur. The model-to-world transformation moves each object from its local coordinate space to world space, aligning all objects within a common 3D coordinate system. The view transformation positions and orients the camera, transforming objects into camera space based on the camera's perspective. Then, the projection transformation is applied, converting 3D coordinates into 2D screen space. This step uses either perspective projection, where distant objects appear smaller, or orthographic projection, where object size remains uniform regardless of distance.

Before these transformations are completed, two key optimization techniques are applied to remove non-visible data, ensuring that only what is necessary is processed. The first is view frustum culling, which discards entire objects that fall outside the camera's view frustum—the truncated pyramid that defines the camera's visible field in 3D space. Objects entirely outside this volume are excluded from further processing, while those partially inside are clipped, improving rendering performance without affecting the visual result. Following this, back-face culling is employed as an additional optimization. This technique removes the faces of objects that are oriented away from the camera, as they are not visible to the user. By discarding these back-facing polygons, the computational load is further reduced, particularly in scenes with complex geometry.

2.1.2.2 Shading and Lighting

With the scene properly transformed into 2D coordinates, the next step is the calculation of light interaction with object surfaces, which determines the final color perceived in each pixel. This stage, known as lighting and shading, varies depending on the rendering approach and is generally expressed through the Rendering Equation [22]:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \int_\Omega f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) L_i(\mathbf{x}, \omega_\mathbf{i})(\mathbf{n} \cdot \omega_\mathbf{i})\, d\omega_\mathbf{i} \tag{2.2}$$

where:

- $L_o(\mathbf{x}, \omega_\mathbf{o})$ is the outgoing radiance at point $\mathbf{x}$ in the direction $\omega_\mathbf{o}$, i.e., the color perceived by the camera.

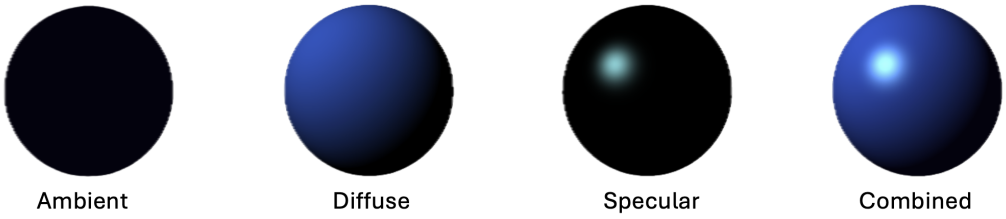


Figure 2.4: Illumination terms of Phong illumination model

- $L_e(\mathbf{x}, \omega_\mathbf{o})$ is the emitted radiance, if the surface emits light.
- $f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o})$ is the Bidirectional Reflectance Distribution Function (BRDF), modeling how light reflects from direction $\omega_\mathbf{i}$ to direction $\omega_\mathbf{o}$.
- $L_i(\mathbf{x}, \omega_\mathbf{i})$ is the incoming radiance from direction $\omega_\mathbf{i}$.
- $\mathbf{n} \cdot \omega_\mathbf{i}$ is the cosine of the angle between the surface normal $\mathbf{n}$ and the incoming direction $\omega_\mathbf{i}$, accounting for how much light strikes the surface.

This equation models both direct illumination (light coming directly from a light source) and indirect illumination (light that has bounced off other surfaces), offering a comprehensive description for the propagation of light in a scene.

In real-time rendering, simplifying assumptions are often made to achieve faster computations. Local illumination models, such as the Phong illumination model, only consider direct light sources and ignore indirect light. The Phong model calculates the total outgoing intensity $L_o(\mathbf{x}, \omega_\mathbf{o})$ at a surface point as the sum of four components: ambient, diffuse, specular, and emitted light:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_a + L_d + L_s + L_e \tag{2.3}$$

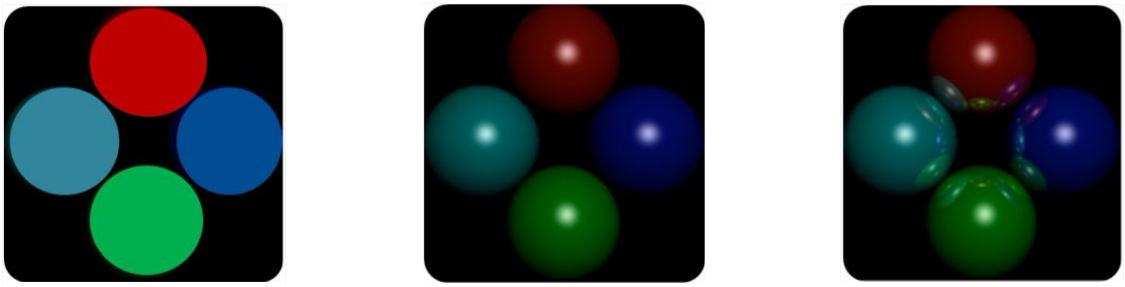
where L_a is the ambient term, representing light scattered uniformly in the environment, L_d is the diffuse reflection, modeling light scattered equally in all directions, calculated using Lambert's cosine law, L_s is the specular reflection, responsible for highlights and L_e is the emitted light, relevant for surfaces that emit light, such as light sources or emissive materials. This formulation of the Phong model directly relates to the Rendering Equation as a simplified case where only direct illumination is considered, and the BRDF f_r is approximated using simple empirical terms.

In contrast, global illumination models solve the full Rendering Equation, including both direct and indirect light. Techniques such as path tracing simulate light bouncing between surfaces, accurately capturing effects like reflections, shadows, and color bleeding. In path tracing, light is traced from the camera through the scene, and the contributions of multiple bounces are accumulated by solving the Monte Carlo approximation of the

Rendering Equation. The final radiance is computed as:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = \frac{1}{N} \sum_{i=1}^{N} f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) L_i(\mathbf{x}, \omega_\mathbf{i})(\mathbf{n} \cdot \omega_\mathbf{i}) \tag{2.4}$$

where N is the number of sampled light paths.



(a) No illumination **(b)** Local illumination **(c)** Global illumination

Figure 2.5: Comparison between different illumination models

2.1.2.3 Rasterization

During rasterization, the geometric primitives (usually triangles) are converted into fragments or pixel data by mapping the 2D projections of 3D objects onto the screen's pixel grid. Each fragment's color is then calculated based on the object's material properties, lighting conditions, and shading model. To ensure that only the visible fragments are rendered, depth testing is applied. A common technique for this is Z-buffering, where each pixel is associated with a depth value (the distance from the camera). As fragments are processed, the Z-buffer stores the depth of the closest fragment at each pixel. If a new fragment is found to be closer than the current one, it replaces the old fragment, ensuring that only the nearest objects are displayed, while fragments of objects that are behind others are discarded. This process effectively handles the occlusion of overlapping objects.

In some cases, additional post-processing effects are applied to enhance visual quality. Techniques like anti-aliasing reduce jagged edges, while bloom adds a glow around bright objects to simulate overexposure. Other effects like motion blur or depth of field are added to simulate cinematic qualities. Finally, tone mapping is applied to compress high dynamic range lighting into a displayable low dynamic range format.

2.1.3 Novel view synthesis

Novel view synthesis (NVS) is a technique that generates new viewpoints of a scene using a limited number of images. This approach creates immersive experiences, allowing users to explore perspectives that were not initially captured. By utilizing the geometric and

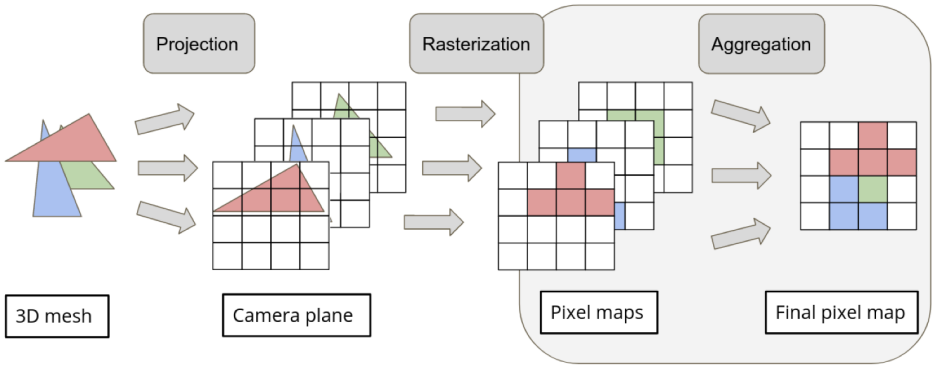


Figure 2.6: Overview of the rendering process

photometric data embedded in the input images, novel view synthesis reconstructs the visual appearance of a scene from these fresh angles. In the field of 3D reconstruction, this technique depends on comprehending the spatial relationships between various viewpoints. By synthesizing new views from the existing data, it enhances the richness and depth of scene representation. From the rendering perspective, novel view synthesis is vital for producing visually compelling images. It employs algorithms that interpolate and blend data from the input images, ensuring consistency in lighting, texture, and color with the original scene. Recent advancements in neural networks have further transformed this field by enabling the learning of complex scene representations.

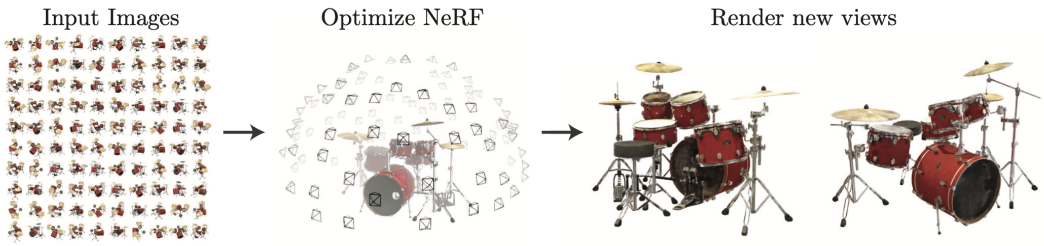


Figure 2.7: NeRF 3.1 is a method that optimizes a continuous 5D neural radiance field representation of a scene from a set of input images. Here, the set of 100 input views of the synthetic Drums scene randomly captured on a surrounding hemisphere is visualized, and then two novel views rendered from the optimized NeRF representation are shown.

2.2 Rendering techniques

In this section an overview of the main rendering approaches and techniques is presented. This classification is not strict since in many practical approaches, different methods can be combined to obtain better results.

2.2.1 Physically based rendering

Physically Based Rendering (PBR) is a rendering approach aimed at creating realistic images by accurately simulating how light interacts with surfaces, based on the physical properties of materials. The primary objective of PBR is to ensure that surfaces respond consistently to lighting across different scenes and viewpoints, producing photo-realistic results. It takes into account material properties like color, roughness, and reflectivity, which influence how light is reflected, refracted, and scattered. Two techniques commonly used in PBR are ray casting and ray tracing.
Ray casting is a simpler method that involves casting rays from the camera into the scene to determine what objects are visible in the final image. Each ray checks for intersections with objects along its path, determining which object is hit first. Ray casting is computationally efficient, as it only checks visibility, but doesn't simulate complex lighting effects like shadows, reflections, or refractions.
Ray tracing, on the other hand, is more comprehensive. It not only determines visibility but also simulates how light behaves as it interacts with surfaces. When a ray hits a surface, secondary rays are cast to calculate reflections, refractions, shadows, and indirect lighting effects. This produces much more realistic images, as it mimics the actual behavior of light in the real world, but at a higher computational cost.

PBR leverages these techniques to model light behavior in a way that faithfully represents real-world phenomena. The interplay of light with materials, guided by physics-based algorithms, allows PBR to deliver scenes with lifelike visual fidelity. This has made PBR widely used in industries like film, gaming, and virtual reality, where realism and material authenticity are key.

2.2.2 Volume rendering

Volumetric representations model objects and scenes not just as surfaces but as volumes filled with materials or empty space. This allows for a more accurate rendering of phenomena like fog, smoke, or translucent materials. Ray-marching is a technique used with volumetric representations to render images by tracing the path of light through a volume [11]. In this method, rays are cast from the camera through the volume, sampling data at regular intervals along their paths. At each step, the algorithm accumulates color and opacity values based on the properties of the volume at that point, resulting in the final image. This process can create realistic views of volumetric effects by simulating how light travels through and interacts with semi-transparent materials. NeRF [33] shares the same spirit of volumetric ray-marching and introduces importance sampling and positional encoding to improve the quality of synthesized images. While providing high-quality results, volumetric ray-marching is computationally expensive, motivating the search for more efficient methods.

2.2.3 Point-based rendering

Point-based rendering is a technique for visualizing 3D scenes using points rather than traditional polygons. This method is particularly effective for rendering complex, unstructured, or sparse geometric data. In volumetric rendering, a significant drawback is the inefficiency caused by sampling points in unoccupied spaces, which contribute little to the final image. In contrast, point-based methods, which focus on occupied regions of the scene, offer a more efficient and precise representation by relying on rasterization rather than stochastic sampling.

Points can be augmented with additional properties like learnable neural descriptors [1] [38], and rendered efficiently [28] [40], but this approach suffers from issues like holes in the rendering or aliasing effects. 3D Gaussian Splatting [24] extends this concept by using anisotropic Gaussians for a more continuous and cohesive representation of the scene.

2.2.4 Neural Rendering

Neural rendering combines ideas from classical computer graphics and machine learning to create algorithms for synthesizing images from real-world observations.
While classical computer graphics starts from physics, by modeling for example geometry, surface properties and cameras, machine learning comes from a data-drive, statistical perspective, i.e., learning from actual real world examples to generate new images. By leveraging large-scale datasets imagery, deep learning models can learn complex mappings from input images to novel views or modifications, addressing both the reconstruction and rendering tasks. Neural rendering incorporates mathematical models of projection and combines them with learned components to create flexible systems capable of controlling scene properties such as geometry, illumination, camera viewpoint, pose, and semantic structure [46].

The applications of neural rendering range from novel view synthesis, semantic photo manipulation, facial and body reenactment, relighting, free-viewpoint video, to the creation of photo-realistic avatars for virtual and augmented reality telepresence.

2.2.5 Differentiable Rendering

Differentiable rendering [23] is a technique that integrates the principles of computer graphics with machine learning by enabling the gradients of rendering parameters to be computed and propagated through the rendering process. Traditional rendering pipelines treat the process of converting a 3D scene into a 2D image as non-differentiable, meaning that small changes in parameters like geometry, lighting, or camera pose cannot be easily related to changes in the output image. Differentiable rendering, however, modifies this pipeline so that these relationships can be calculated through backpropagation,

enabling optimization tasks where rendering parameters need to be learned from data. This capability is exploited in applications like inverse rendering, where the goal is to recover scene properties (such as material, light, or shape) from a 2D image.

Differentiable rendering allows for fine-tuning scene parameters based on how closely the rendered output matches target images. Techniques like NeRF leverage differentiable rendering to generate high-quality 3D scenes from 2D images by learning the underlying volumetric structures of the scene. Furthermore, differentiable rendering plays a role in training deep learning models for tasks like image synthesis, reconstruction, and even reinforcement learning, where environmental feedback is rendered and optimized for through differentiable pipelines. The development of techniques like 3D Gaussian Splatting has further enhanced the efficiency of real-time rendering applications by enabling the gradients to be computed over the rendering process, thus bridging the gap between traditional rendering and modern machine learning approaches.

2.3 Real time Rendering

Real-time rendering refers to the process of generating images or animations quickly enough to allow for interactive and dynamic viewing experiences. This technology is essential for applications where responsiveness and immersion are critical, such as video games, Virtual Reality (VR), Augmented Reality (AR), simulations, and interactive visualizations. A key aspect of real-time rendering is achieving a high frame rate (measured in Frames Per Second, FPS), which ensures smooth visual feedback to the user. Maintaining a frame rate of at least 30 FPS is generally considered the minimum for a fluid experience, although 60 FPS or higher is often required in more demanding applications like VR, where low latency and fast responsiveness are crucial to avoid motion sickness. The rate of frames per second makes it seem as though the user is walking through a realistic world rather than interacting with a computer program, and any noticeable drop in FPS can break the sense of immersion.

The most significant differences between offline and real-time rendering are their speed and amount of interactivity. Offline rendering refers to the creation of static images or videos in advance and saving them for further alteration. This method allows for the synthesis of a highly polished finished product but allows very little interactivity and typically takes anywhere from minutes to hours to render. Real-time rendering, on the other hand, must produce frames within milliseconds to achieve interactive performance, typically targeting under 16.67 milliseconds per frame to grant 60 FPS to the viewer. This allows the user to create highly interactive, easily manipulated 3D simulations that render in less than a second.

2.3.1 eXtended Reality

A key application of real-time rendering lies in eXtended Reality (XR), where interactivity and immersion are paramount. Extended Reality refers to a spectrum of immersive technologies that blend the real and digital worlds to create interactive and dynamic environments. These technologies enable real-time interaction with virtual elements, enhancing sensory perception and introducing new forms of engagement. XR encompasses various immersive technologies associated with different types of realities, including VR, AR, Mixed Reality (MR), with XR often serving as a broad, umbrella term for these technologies. In fact, the *X* in XR represents more than just *extended*, it can be interpreted as a variable symbolizing future innovations that will fall under this evolving technological domain. The recent spread of XR applications on mobile and wearable devices has introduced new challenges for designers and developers.

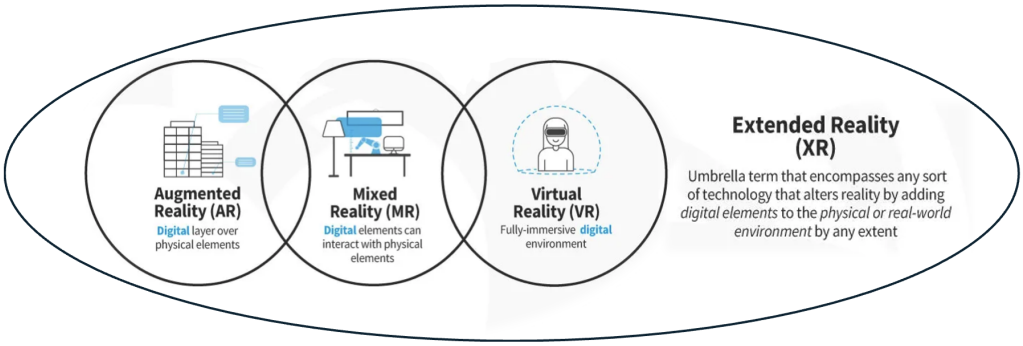


Figure 2.8: eXtended Reality

2.3.2 Problem Statement

Rendering in extended reality presents significant challenges in terms of computational efficiency, visual fidelity, and system latency. XR applications require rendering 3D scenes that adapt instantaneously to the user's head and body movements, creating an highly immersive and responsive experience that demands low latency and high frame rates. This requires accurate alignment of the virtual scene with the user's real-world position and orientation, such process demands substantial computational resources, particularly in stereo rendering for virtual reality, where the scene must be rendered separately for each eye.

A major issue in real-time XR rendering is balancing visual fidelity with performance constraints. High-quality rendering with detailed textures, shadows, and lighting effects is computationally expensive, but reducing these elements reduces the user's sense of immersion. Even slight delays or visual inconsistencies, such as frame rate drops or mismatched head movement tracking, can cause discomfort or "cybersickness." Ensuring that the rendered scene aligns perfectly with the user's movements in real-time is critical, and failing to do so can result in a disorienting experience. Additionally, unlike desktop computers with powerful GPUs and extensive cooling systems, XR headsets must balance processing power, heat dissipation, and battery life. These constraints mean that optimization techniques, such as Level of Detail management, foveated rendering, and variable rate shading, must be employed to reduce the computational load without sacrificing visual quality. These methods attempt to simplify or reduce detail in less important areas of the scene, focusing rendering power on areas directly in the user's field of view, helping in balance the demands of high-quality visuals with the need for low latency and high performance.

Various techniques have been employed to adjust the LODs based on user proximity and interaction [3], [18]. Previous research has focused on adapting cognitive load [10], predicting user actions to optimize training experiences [49], or minimizing transmitted information [25], while only a few recent attempts have been made using Deep Neural

Networks [5], [6]. On the other hand, many techniques use 3D reconstruction technologies to improve optimized representations of 3D models. Technologies such as Neural Radiance Fields (NeRF) [33] and 3D Gaussian Splatting (3DGS) [24] support the development of more efficient and immersive XR applications by building implicit representations of the 3D scene that can be rendered later on demand from a specific point of view to obtain the desired perspective.

2.3.3 Visual Quality Adaptation

Visual quality adaptation aims at balancing user experience and hardware constraints, especially in resource-intensive environments like extended reality. Adaptive rendering techniques prioritize visual fidelity in areas where the user's gaze is focused, reducing the quality in peripheral regions. This allows for high performance while maintaining the immersive experience fundamental for XR applications.

The problem of quality maximization strategy under bandwidth constraints first appeared in video transmission [36]. Several HTTP Adaptive Streaming (HAS) strategies propose packet-dropping strategies to tune the transmission stream while minimizing the quality decrement [43], [45], [21], [35], [50]. Most of these strategies rely on linear programming solvers on simplified parametric rate and quality models. Later, deep learning solutions have been investigated to address this problem in the video domain [13], [31], [12] and allowed the extension to XR applications. 3D scene implicit representation has opened new possibilities in this field to synthesize and reproduce selectively 3D scene parts with variable LOD. This approach allows for real-time adaptations based on the user's focus, ensuring that resources are allocated efficiently without sacrificing the overall visual quality.

2.3.3.1 Level of Detail

Level of Detail is a foundational concept in computer graphics, particularly in real-time rendering, aimed at improving performance by adjusting the complexity of 3D models based on their significance in a given scene. LOD techniques work by reducing the geometric complexity of objects, often based on their distance from the viewer, to minimize the computational load without sacrificing the overall visual fidelity of the scene. For example, objects further from the viewer are rendered with fewer polygons or vertices, while those closer are rendered with higher detail. This approach allows for maintaining high frame rates, especially critical in XR environments where real-time responsiveness is essential to avoid latency and motion sickness.

The concept of LOD can be traced back to early discussions in computer graphics, such as [41] work on hierarchical geometric models for visible surface algorithms. Since then, the technique has evolved significantly, incorporating various approaches such as mesh decimation, where non-essential elements of a 3D mesh are removed to maintain a budget

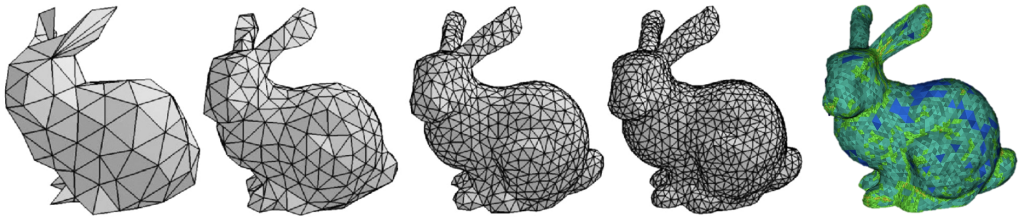


Figure 2.9: The Stanford Bunny model at different LODs, the last one on the right, visualized with colors, is the model with adaptive LOD, i.e., different level of detail in different areas of the model based on the complexity of the model in such areas

of faces and vertices while preserving visual quality. More modern methods focus on optimizing perceptual metrics and addressing challenges in smooth transitions between different LODs, which is critical to avoid visual artifacts during real-time rendering. In the context of XR, by dynamically selecting and transitioning between different LODs based on user movement and scene importance, systems can balance the high demand for visual fidelity with limited processing power.

3

State Of the Art

The state of the art in rendering has seen significant advancements with the development of NeRF [33] and 3D Gaussian Splatting [24], two innovative methods that are revolutionizing how complex 3D scenes are represented and rendered. NeRF, introduced in 2020, utilizes neural networks to synthesize highly detailed 3D scenes by learning a continuous volumetric scene function from a set of 2D images. By capturing the way light interacts with objects at each point in space, NeRF can render novel views of a scene with photorealistic quality, even from sparse input data. Its differentiable nature allows NeRF to optimize scene parameters through gradient-based methods, making it highly effective in applications such as view synthesis, 3D reconstruction, and inverse rendering.

Another innovative approach is 3D Gaussian Splatting, proposed in the middle of year 2023. Its introduction represents a fundamental shift in how we approach scene representation and rendering in computer vision and graphics. Unlike previous method relying on implicit, coordinate-based models, 3DGS employs millions of learnable 3D Gaussians, offering an explicit scene representation and a differentiable rendering algorithm. This explicit nature of 3DGS facilitates real-time rendering capabilities and unprecedented levels of control and editability. Unlike NeRF, which can be computationally expensive due to its reliance on neural networks, 3DGS provides a more lightweight solution while preserving differentiability, making it suitable for real-time applications such as XR.

Both NeRF and 3DGS exemplify the shift toward using neural and probabilistic representations for 3D rendering, which offer significant advantages in terms of flexibility, quality, and efficiency over traditional methods. These techniques have opened new avenues for photorealistic rendering, dynamic scene interaction, and scalable solutions for resource-constrained environments.

3.1 NeRF

Neural Radiance Field [33] models are novel view synthesis methods which use volume rendering with continuous neural scene representation via Multi Layer Perceptrons (MLPs) to learn the geometry and lighting of a 3D scene.
In its basic form, a NeRF model represents three-dimensional scenes as an implicit radiance field approximated by a neural network. The radiance field describes both the color and volume density at any point in the scene for any viewing direction. Formally, the radiance field is modeled as:

$$F(x, \theta, \phi) \rightarrow (c, \sigma) \tag{3.1}$$

whose input is a 3D location $x = (x, y, z)$ and 2D viewing direction (θ, ϕ), and whose output is an emitted color $c = (r, g, b)$ represents color and volume density σ. In practice, direction is expressed as a 3D Cartesian unit vector $d = (d_x, d_y, d_z)$.
This 5D function is approximated by a Multi-Layer Perceptron network

$$F_\Theta(x, d) \rightarrow (c, \sigma) \tag{3.2}$$

and its weights Θ are optimized to map from each input 5D coordinate to its corresponding volume density and directional emitted color.
To ensure consistency across multiple views, the NeRF model constrains the volume density σ to be independent of the viewing direction, while allowing the color c to vary based on both the viewing direction and the spatial coordinates. In the original NeRF model, this is achieved using a two-stage MLP. The first stage processes the spatial coordinates x and outputs the volume density σ along with a high-dimensional feature vector. The second stage concatenates this feature vector with the viewing direction d and passes it through an additional MLP to predict the color c.

In general, novel view synthesis with a trained NeRF model involves the following steps. For each pixel of the synthesized image, camera rays are cast through the scene, and a set of sampling points is generated along each ray. At each sampling point, the local color and density are computed using the viewing direction and sampling location with NeRF's MLP. The final image is then produced using volume rendering based on these computed colors and densities.

Given the volume density and color functions for the scene, volume rendering is used to determine the color $C(r)$ of any camera ray $r(t) = o + td$, where o is the camera position and d is the viewing direction:

$$C(r) = \int_{t_1}^{t_2} T(t)\sigma(r(t))c(r(t), d)\, dt \tag{3.3}$$

where $\sigma(r(t))$ is the volume density and $c(r(t), d)$ is the color at point $r(t)$, while dt

represents the differential distance traveled along the ray. The accumulated transmittance $T(t)$, which represents the probability of the ray traveling from t_1 to t_2 without being blocked, is given by:

$$T(t) = \exp \int_{t_1}^{t_2} \sigma(r(s))\, ds \tag{3.4}$$

To generate novel views, camera rays are traced through each pixel of the desired image. The integral can be computed numerically, and the original implementation used stratified sampling, where the ray is divided into N equally spaced bins, with samples drawn from each bin. The rendered color $\hat{C}(r)$ is approximated as:

$$\hat{C}(r) = \sum_{i=1}^{N} a_i T_i c_i, \qquad \text{where} \qquad T_i = \exp - \sum_{j=1}^{i-1} \sigma_i \delta_i \tag{3.5}$$

Here, δ_i is the distance between samples i and $i+1$, (σ_i, c_i) are the density and color at the sampling point, and α_i is the transparency at sample point i, computed as $\alpha_i = 1 - exp(-\sigma_i \delta_i)$. Finally, the expected depth of the ray can be calculated using the accumulated transmittance as:

$$d(r) = \int_{t_1}^{t_2} T(t) \cdot \sigma(r(t)) \cdot t\, dt \tag{3.6}$$

Similarly to (3.5), this can be approximated as

$$\hat{D}(r) = \sum_{i=1}^{N} a_i t_i T_i \tag{3.7}$$

For each pixel, a square error photometric loss is used to optimize the MLP parameters. Over the entire image, this is given by

$$L = \sum_{r \in R} ||\hat{C}(r) - C_{gt}(r)||_2^2 \tag{3.8}$$

where $C_{gt}(r)$ is the ground truth color of the training image's pixel associated to r, and R is the batch of rays associated to the to-be-synthesizes image. NeRF models often employ positional encoding to improve fine detail reconstruction in the rendered views.

Despite NeRF's significant achievements, it still faces limitations, particularly regarding slow training and rendering speeds. Various follow-up techniques [14] [37] aim to enhance rendering quality or accelerate training using methods like hashing [34] or sparse 3D grids [55]. However, these methods still struggle to accurately represent empty spaces and encounter image quality constraints due to structured grids, which can substantially slow down rendering performance.

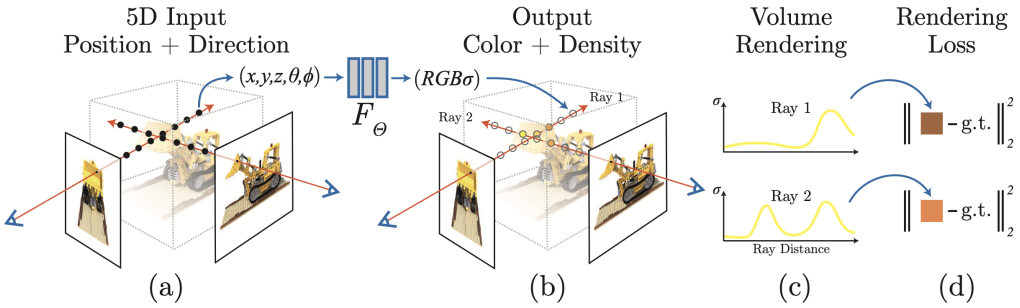


Figure 3.1: Overview of neural radiance field scene representation and the differentiable rendering process. Images are synthesized by sampling 5D coordinates (location and viewing direction) along camera rays (a). These coordinates are then fed into a MLP to produce color values and volume density (b). Volume rendering techniques are subsequently used to composite these values into a final image (c). This rendering function is differentiable, allowing for the optimization of scene representation by minimizing the residual error between the synthesized images and the ground truth images (d).

3.1.1 Applications and limitations

NeRF has become a popular tool for synthesizing photorealistic 3D scenes from 2D images. However, some challenges and limitations still remain. One of the major drawbacks of NeRF is its memory requirements, which can be a significant bottleneck for practical applications. The high memory consumption of NeRF is due to the need to store the neural network parameters, which can be challenging for large-scale scenes or high-resolution images. The computational expense of NeRF is also a limitation, particularly during training. Evaluating the neural network for each ray in the scene is time-consuming, and techniques proposed to accelerate the training of NeRF may come at the cost of decreased performance. Another limitation of NeRF is its inability to handle dynamic scenes or moving objects. NeRF only learns a single representation of the scene, which needs to be improved to address changes in the scene over time. Attempts to extend NeRF to handle dynamic scenes are an active research area. In addition, NeRF is sensitive to the quality of the input images, as it relies on accurate depth and camera pose estimates, which may be challenging to obtain in practice, especially for complex scenes. Finally, NeRF needs more interpretability, as there is a need to understand why particular views are synthesized correctly or incorrectly. Developing techniques for understanding and visualizing learned representations of NeRF is an important direction for future research. Furthermore, NeRF's slow inference speed, reliance on accurate pose estimation and multiple views, and limited effectiveness in scenarios with sparse views or poor camera calibration are additional limitations that must be addressed.

3.2 3D Gaussian Splatting

3D Gaussian Splatting [24] is an approach that models scene geometry using 3D Gaussians instead of relying on traditional point-based or surface-based representations. The key motivation behind using 3D Gaussians is their ability to represent the scene in a continuous, differentiable manner that can be optimized for rendering.

Each 3D Gaussian is defined by a position (mean) $\mu \in \mathbb{R}^3$, a covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$, a color c represented by spherical harmonics for view-dependent appearance and an opacity $\alpha \in \mathbb{R}$. All the properties are learnable and optimized through back-propagation. The 3D Gaussian function is defined as follows:

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{3.9}$$

where x is a point in 3D space. The covariance matrix Σ is anisotropic, determining both the shape and orientation of the Gaussian.

An anisotropic covariance matrix represents an ellipsoidal Gaussian, where the variance differs along different axes. This leads to a Gaussian that is stretched or compressed in certain directions, forming an ellipsoid. The shape and orientation of the ellipsoid are dictated by the eigenvalues and eigenvectors of the covariance matrix. In contrast, an isotropic covariance matrix represents a spherical Gaussian, meaning that the variance is the same in all directions. In other words, the Gaussian is perfectly symmetric, and the spread of the points from the center is uniform along every axis. This occurs when the covariance matrix is proportional to the identity matrix, indicating no directional preference in the data distribution. The anisotropy allows the Gaussian to capture more complex, directionally dependent structures in the data.

For rendering, the 3D Gaussians (ellipsoids) in 3D space are projected into the 2D image space (ellipses) by transforming the covariance matrix Σ using the viewing transformation. This projection process essentially "splatters" each 3D Gaussian onto the image plane, resulting in a 2D elliptical footprint that can be blended smoothly with other splats using α blending techniques. This allows for fast rendering since each splat can be rendered independently, making it suitable for real-time applications.
The covariance matrix Σ is optimized directly in world space, but to ensure physical validity (i.e., that the matrix remains positive semi-definite), the optimization is performed on its underlying components. Specifically, the covariance matrix is decomposed into a scaling matrix S and a rotation matrix R:

$$\Sigma = RSS^TR^T \tag{3.10}$$

where S controls the anisotropic scaling of the Gaussian (i.e., the size of the ellipsoid in different directions) and R defines the orientation of the ellipsoid. The scaling is rep-

resented as a 3D vector s, and the rotation is represented using a quaternion q. This representation ensures that during optimization, the covariance matrix remains valid, as quaternions are normalized to maintain proper rotations. By optimizing these parameters, the 3D Gaussians can adapt to the underlying geometry of the scene, capturing shapes and features in a compact and expressive form. This approach allows for efficient optimization using gradient descent while maintaining the differentiable nature of the volumetric representation, enabling high-quality novel view synthesis from sparse inputs.

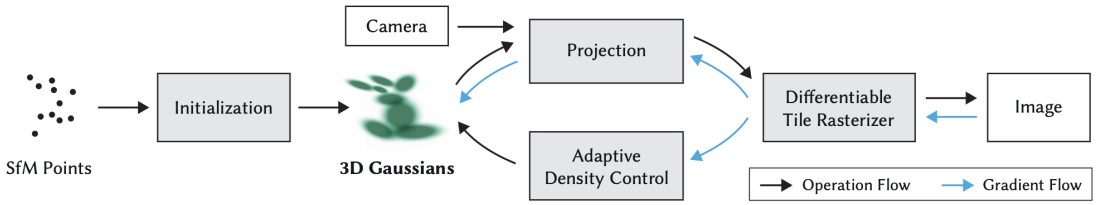


Figure 3.2: 3DGS optimization starts with the sparse SfM point cloud and creates a set of 3D Gaussians. Then the density of this set of Gaussians is optimized and adaptively controlled. During optimization fast tile-based renderer is used, allowing competitive training times. Once trained, our renderer allows real-time navigation for a wide variety of scenes [24].

3.2.1 Optimization

In the optimization process, a dense set of 3D Gaussians is iteratively refined to precisely model the scene's geometry and appearance for free-viewpoint rendering. This involves two primary challenges. First, the properties of each Gaussian must be optimized through differentiable rendering to match the textures of the scene. Second, the number of 3D Gaussians that can represent a given scene well is unknown in advance. A promising approach is to allow the neural network to automatically learn the density of these Gaussians. The two procedures are interleaved within the optimization workflow.

As the optimization progresses, the algorithm creates new Gaussians where geometry is underrepresented and also modifies or removes Gaussians when the geometry is incorrectly placed. This flexibility ensures that the representation is dynamic and can accurately adapt to the scene as the optimization evolves. The quality and precision of the covariance matrices of the Gaussians are important to maintaining a compact representation. In large homogeneous regions, for instance, a few large anisotropic Gaussians can capture the structure effectively, whereas more detailed areas may require a higher density of smaller Gaussians.

To achieve this optimization, Stochastic Gradient Descent (SGD) is utilized, leveraging GPU-accelerated frameworks to ensure computational efficiency. To regulate the opacity α of the Gaussians, a sigmoid activation function is employed, which ensures smooth gradients and keeps α within the range [0, 1). Similarly, an exponential activation function is used for the covariance scale to maintain positive values and smooth optimization. Initially, the covariance matrix of each Gaussian is estimated as an isotropic Gaussian, with the axes determined by the average distance to the three nearest points. This initialization helps establish a reasonable starting point for the optimization of the 3D Gaussians.

The loss function balances ℓ_1 loss, which measures pixel-wise differences between the rendered and ground truth views, with a D-SSIM term, a differentiated version of Structural Similarity Index (SSIM) that enhances the structural similarity between the rendered views and the ground truth views from the training dataset:

$$\ell = (1 - \lambda)\ell_1 + \lambda\ell_{D-SSIM} \tag{3.11}$$

where λ is a weighting parameter. An exponential decay learning schedule for the positions of the Gaussians is followed. This scheduling ensures that the positions of the Gaussians are progressively refined throughout the optimization.

3.2.2 Adaptive Control of Gaussians

3DGS starts with the initial set of sparse points from SfM. Afterwards, point densification and pruning are adopted to control the density of 3D Gaussians.

The process iteratively adjusts the Gaussian representation. After an initial optimization warm-up phase, Gaussians are densified every 100 iterations to refine the scene. Simultaneously, Gaussians with very low opacity are culled—those with α below a threshold are considered effectively transparent and are removed. This approach ensures that only the Gaussians contributing meaningfully to the scene remain, preventing an unnecessary increase in complexity.
The method specifically targets two types of regions for densification:

- under-reconstructed areas, where key geometric details are missing,
- over-reconstructed areas, where Gaussians cover too large a region.

Both cases are characterized by high positional gradients in view-space, signaling regions that the optimization process is attempting to correct. These gradients serve as indicators of regions where the Gaussians need to be adapted.
For under-reconstructed regions, small Gaussians are duplicated and moved in the direction of the positional gradient to cover new geometry. This cloning process effectively increases the total volume of Gaussians in these regions, allowing the method to capture additional scene details. On the other hand, large Gaussians in over-reconstructed areas are split into two smaller Gaussians, each scaled down by a specific factor. The splitting

process does not increase the total volume but instead increases the number of Gaussians, providing a more fine-grained representation in regions of high variance.

(a) Under-Reconstraction **(b)** Over-Reconstraction

Figure 3.3: Adaptive Gaussian densification scheme. (a) When small-scale geometry (black outline) is insufficiently covered, the respective Gaussian are cloned. (b) If small-scale geometry is represented by one large splat, it is splitted in two [24].

The method also incorporates mechanisms to prevent excessive growth in the number of Gaussians, which can occur due to optimization issues, such as "floaters" near the input cameras. These floaters, which are unneeded Gaussians that cluster in areas with high view-space positional gradients, can lead to unjustified increases in Gaussian density. To mitigate this, the α value of all Gaussians is periodically reset to a value close to zero every $N = 3000$ iterations. This reset allows the optimization process to increase α only for those Gaussians that are truly necessary, while enabling the removal of any Gaussians whose α remains below a threshold.

In addition to density control, the method handles the dynamic scaling of Gaussians. Over the course of optimization, Gaussians may grow or shrink significantly, sometimes leading to excessive overlap between neighboring Gaussians. To address this, the algorithm removes Gaussians that become too large in world-space or have an overly large footprint in view-space. This ensures that the total number remains manageable and the representation stays efficient without resorting to complex space compaction, warping, or projection strategies often used in other volumetric methods.

3.2.3 Rasterization

Rasterization refers to the process of converting 3D Gaussian splats into a 2D image by projecting them onto a screen and blending their contributions according to their depth, size, and transparency. To achieve fast rendering, the rasterization process is divided into several key steps.

First, the screen is subdivided into tiles (typically 16×16 pixels), allowing parallel processing of Gaussians within each tile. This tiling strategy ensures that only the relevant Gaussians affecting a specific region of the screen are considered, minimizing computational overhead and enhancing scalability for large scenes. Each Gaussian is culled against the view frustum and tiles, meaning only those Gaussians whose 3D projection

intersects the viewable area are processed. This culling ensures that the Gaussians outside the camera's field of view or in positions where they would contribute little to the final image are discarded early in the pipeline, improving efficiency.

Once the relevant Gaussians are identified, they are sorted based on their depth in view space. This sorting step is important for accurate α-blending, which is the process of correctly layering transparent splats in the image according to their distance from the camera. Sorting the Gaussians by depth ensures that those closer to the camera are rendered in front of those further away, allowing for a realistic blending of splats where semi-transparent regions overlap. The sorting is done using a GPU-based Radix sort algorithm, which minimizes the time complexity of handling large numbers of Gaussians while avoiding the need for per-pixel sorting, a bottleneck in previous approaches.

Once sorted, each tile is processed by a separate thread block. The Gaussians affecting each tile are loaded into shared memory, where their contributions to each pixel are computed. For each pixel, the color and opacity α are accumulated by traversing the sorted list of Gaussians from front to back. The traversal continues until the α value for the pixel reaches saturation ($\alpha = 1$), at which point no further Gaussians contribute to the pixel's color. For a color at a pixel i , with n indexing the N Gaussians involved in that pixel, the color C_i is calculated as:

$$C_i = \sum_{n=1}^{N} c_n \alpha_n' T_n \qquad \text{where} \qquad T_n = \prod_{m<n} (1 - \alpha_m') \tag{3.12}$$

In this formula, T_n is the contribution from all Gaussians before n, c_n is the learned color of the n-th Gaussian, and α_n' is the final opacity, computed using learned opacity α_n the Gaussian's 2D covariance matrix $\Sigma' \in \mathbb{R}^{2\times2}$. The opacity is calculated as:

$$\alpha_n' = \alpha_n \cdot \exp(-\sigma_n), \quad \sigma_n = \frac{1}{2} \Delta_n^T \Sigma'^{-1} \Delta_n \tag{3.13}$$

where $\Delta_n \in \mathbb{R}^2$ represents the offset between the pixel center and the center of the 2D Gaussian. This ensures that each Gaussian's contribution to the pixel is based on its distance from the camera, its size, and its proximity to the pixel center.

This approach maximizes parallelism, as multiple pixels can be processed simultaneously within a tile, and threads can collaboratively handle the data sharing and processing of Gaussians.

The rasterization process is fully differentiable, meaning it allows the computation of gradients necessary for optimization. This differentiability is essential for tasks such as novel view synthesis, where the 3D Gaussians need to be adjusted to minimize a loss function during training. During the backward pass, the same sorted list of Gaussians is used to compute the gradients. By traversing the list in reverse order (back-to-front), the algorithm efficiently computes how each Gaussian contributed to the final opacity

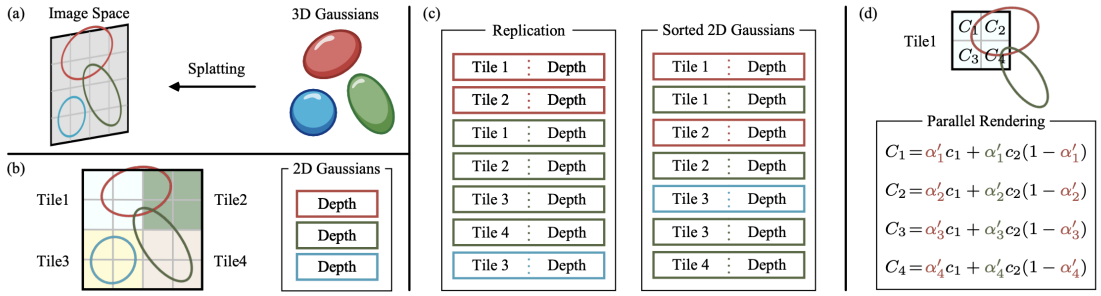


Figure 3.4: An illustration of the forward process of 3D GS. (a) The splatting step projects 3D Gaussians into image space. (b) 3D GS divides the image into multiple non-overlapping patches, i.e., tiles. (c) 3D GS replicates the Gaussians which cover several tiles, assigning each copy an identifier, i.e., a tile ID. (d) By rendering the sorted Gaussians, we can obtain all pixels within the tile. The computational workflows for pixels and tiles are independent and can be done in parallel. [7]

and color of each pixel. This gradient computation does not require storing a large list of intermediate values, which would consume excessive memory; instead, the accumulated opacity is stored during the forward pass, and the necessary coefficients for the gradients are computed dynamically during the backward traversal.

3.3 Opportunities and Limitations

3D Gaussian Splatting is an emerging technique in computer graphics and vision, offering numerous opportunities for efficient rendering and 3D reconstruction. This method stands out for its ability to use Gaussian distributions to represent volumes and surfaces, providing significant computational efficiency. However, several limitations and challenges hinder its broader application. One of the primary issues involves the presence of floating elements in the 3D rendered space, often arising from background interference in the images. These elements, known as "floats", compromise the visual fidelity of the rendering. While opacity thresholds have been proposed to mitigate this issue, more advanced strategies are needed to anchor these elements to nearby surfaces, improving spatial coherence and enhancing overall image quality.

Another challenge lies in balancing rendering speed and reconstruction quality. While 3DGS allows for efficient mesh reconstruction, these meshes are often discontinuous. For instance, opacity-based approaches, such as SuGaR [17], generate 3D Gaussians around mesh surfaces, but at the cost of rendering quality. Similarly, methods like 2D GS [20] compress 3D volumes into 2D Gaussian disks, improving reconstruction quality but losing some volumetric information. A promising direction for future research is to improve the combination of 3DGS with advanced representations, such as signed distance fields (SDF), to achieve smoother and more continuous surfaces, although this comes at the

expense of increased reconstruction times.

The integration of advanced lighting decomposition techniques presents another challenge for 3DGS, as current methods struggle with scenes featuring indistinct boundaries. This limitation often necessitates the use of object masks during optimization. Due to the particle-like properties of the point clouds generated by 3DGS, which differ from traditional surface points, problems arise in accurately reproducing reflections and anisotropic effects. Incorporating multi-view stereo techniques into the optimization process could improve geometric accuracy and the realistic rendering of surfaces.

In terms of real-time rendering, progress has been made with methods like Scaffold-GS [32], which introduces anchor points from a sparse grid to distribute local Gaussians and improve rendering speed. However, the use of a uniform grid limits the method's adaptability, indicating the need for adaptive representations, such as octrees, to handle more complex and detailed scenes. Octree-based representations offer flexibility in dividing complex regions into smaller grids for detailed processing, but they still require manual adjustments. Scaling these methods to handle large and complex environments, such as urban landscapes, will require further advancements.

Another promising research area involves optimizing 3DGS for few-shot 3D scene generation, but this also presents significant challenges. The success of these techniques heavily depends on the accuracy of monocular depth estimation models, which can vary considerably across different data domains. Additionally, the reliance on fitting to COLMAP [39] points introduces a critical dependency on the performance of the underlying system, complicating optimization in areas with little texture or complex geometries.

Another important development direction involves integrating 3D Gaussian Splatting with large foundation models, as recent research has demonstrated. The incorporation of large-scale language and vision models can enhance 3D scene understanding, improving not only segmentation but also language-guided generation and manipulation of 3D scenes. Furthermore, combining 3DGS with other techniques, such as perception models in autonomous vehicles or large-scale reconstruction models, represents a promising research direction [7].

3.4 Applications

3D Gaussian Splatting is a rapidly evolving field that has opened up new opportunities across a wide range of applications.

- Robotics
  - Simultaneous Localization and Mapping (SLAM): Its ability to represent scenes efficiently and allow fast rendering makes it a suitable choice for robotic systems that require spatial awareness. For instance, GS-SLAM [54] uses 3DGS to achieve real-time tracking and mapping, outperforming alternative methods in terms of speed and efficiency.
  - Navigation: Scene representation based on GS can be used for robotic navigation tasks, allowing robots to plan paths and navigate complex environments. Its ability to handle dynamic scenes and deformations makes it suitable for navigation in real-world scenarios. GaussNav [29], for example, uses 3DGS maps for instance image navigation, showing its potential to guide robots in complex environments.
  - Manipulation: The manipulability of 3DGS makes it valuable for robotic manipulation applications. Robots can interact with objects in a simulated environment represented using 3DGS, enabling the development and testing of manipulation algorithms. The possibility of incorporating physical simulations within 3DGS environments further enhances its suitability for this domain, as demonstrated by PhysGaussian [53], which uses 3D-GS for physics-based simulation tasks.

- Scene reconstruction
  - Large-scale Scene Acquisition: 3DGS's ability to handle large datasets and efficiently render complex scenes makes it suitable for large-scale scene reconstruction tasks. The methods discussed in the sources highlight its use in the reconstruction of urban environments, where it can generate detailed representations of buildings, streets, and other urban objects from real-world data.
  - Dynamic Scene Reconstruction: Advances in 3DGS algorithms have led to its application in the reconstruction of dynamic scenes involving moving objects or temporal deformations. Methods like Gaussian-Flow [15] attempt to capture scene dynamics by approximating the 4D space-time volume, opening up possibilities for reconstructing complex, dynamic environments.
  - Sparse-view Reconstruction: 3DGS has shown promising results in reconstructing scenes from sparse views, which is particularly advantageous in scenarios where dense data acquisition is challenging. By leveraging prior information or employing deep learning techniques, GS-based methods can generate plausible 3D models even with limited image input, expanding its applicability to practical scenarios.

- AI-Generated Content (AIGC)
  - Text-based 3D Asset Generation: 3DGS has shown promise in generating 3D assets from text prompts. Methods like DreamGaussian [44] leverage 3DGS to generate photorealistic 3D assets from textual descriptions, demonstrating its potential for tasks like virtual design and animation.
  - Novel View Synthesis: The real-time rendering capabilities of 3DGS make it well-suited for novel view synthesis tasks. Given multi-view image inputs, 3DGS can generate new views of the scene from arbitrary positions, enabling interactive and immersive experiences in applications like VR and mixed reality.
  - Scene Editing: 3DGS facilitates efficient scene editing due to its explicit representation. Objects within the scene can be manipulated, moved, or removed easily, and new views reflecting the changes can be rendered in real-time, making it valuable for virtual design tools and content creation.

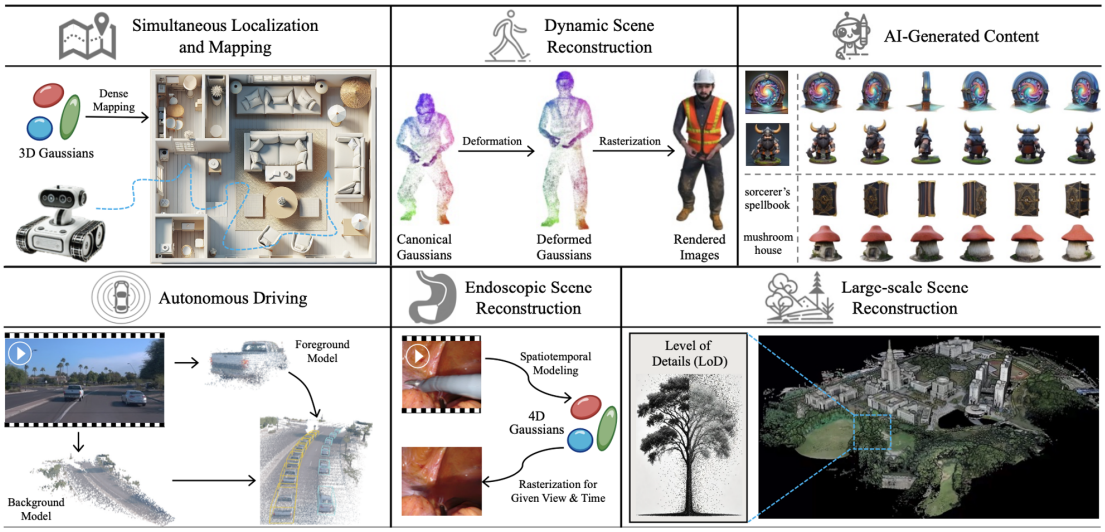


Figure 3.5: Typical applications that have been benefited from 3DGS. [7]

- MEDICAL IMAGING Researchers have explored the use of 3DGS for the reconstruction and visualization of endoscopic scenes. Its ability to handle sparse data and manage deformations makes it suitable for modeling human organs and tissues.
- AUTONOMOUS DRIVING 3DGS methods have been applied to autonomous driving, particularly for tasks like scene reconstruction, pose estimation, and path planning. Its ability to create dense and precise representations of the environment makes it a valuable tool for autonomous driving systems.

4

Dataset Analysis

4.1 Datasets

In this thesis, the scenes analyzed are from the Mip-NeRF360 [2] and Deep Blending [19] datasets. These datasets were also used for the evaluation of 3DGS in [24].

Mip-NeRF360 The Mip-NeRF360 dataset [2] is designed for high-quality scene reconstruction and novel view synthesis, particularly in large-scale, unbounded environments. This dataset provides comprehensive multi-view images along with accurate camera poses. The dataset is composed of 9 scenes (5 outdoors and 4 indoors) each containing a complex central object or area and a detailed background. For each scene, between 100 and 330 images are captured. During capture, camera exposure settings are fixed, lighting variation minimized, and moving objects avoided, to prevent photometric variation between images of the same scene. To further limit color harmonization issues, the outdoor scene are captured when the sky was overcast, making sure that the camera operator cast soft shadows that minimally affected the illumination in the scene. For the indoor scenes, large diffuse light sources (e.g., daylight reflecting off white walls) were used, avoiding any shadows cast onto the scene. Camera poses were estimated using COLMAP [39].

Deep Blending Deep Blending [19] dataset includes 19 scenes captured with a stereo camera rig and reconstructed using COLMAP [39] and Reality Capture. The Deep Blending dataset is specifically designed to facilitate the seamless blending of rendered content with real-world imagery. The dataset enables the analysis of photorealistic transitions between objects and their surroundings, ensuring that the blending maintains consistency in lighting, texture, and geometry. Deep Blending is particularly useful for tasks where rendered objects must be inserted into real scenes or where patches of images from different sources need to be merged cohesively.

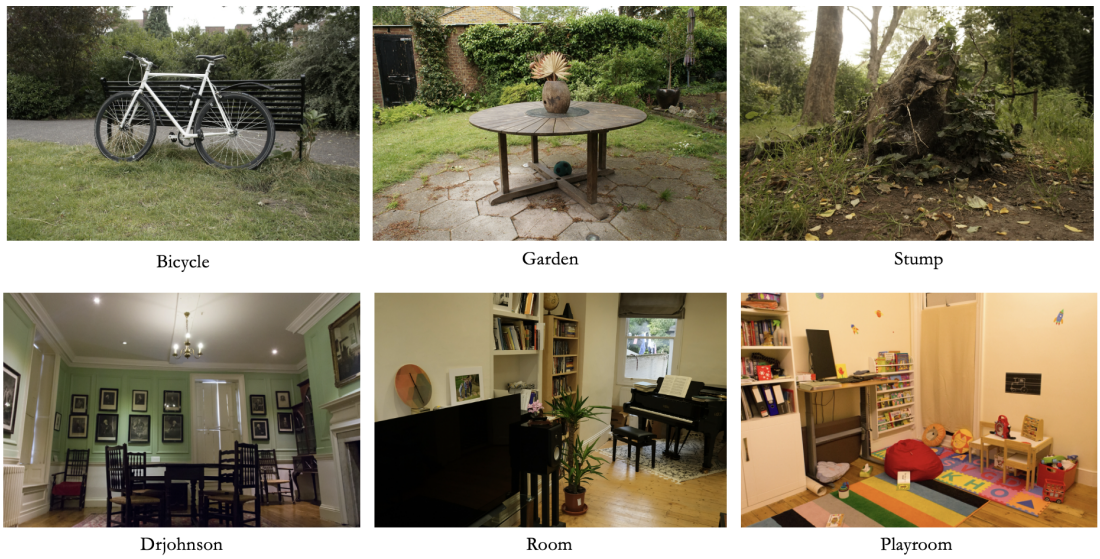


Figure 4.1: Selected scenes, in the top row the outdoor scenes, in the bottom row the indoor scenes

4.1.1 Selected scenes

The scenes selected for analysis are shown in Figure 4.1. This selection was based on the similarity of elements in specific scenes. Three outdoor scenes — `Bicycle`, `Stump`, and `Garden` — were chosen, featuring elements such as grass, plants, sky, and a few other objects. Additionally, indoor scenes — `Drjohnson`, `Room`, and `Playroom` — were selected, characterized by furniture items like sofas, chairs, tables, paintings, and various common and uncommon household objects. These two types of scenes present distinct challenges and enable a comprehensive and diverse analysis. While the indoor scenes contain more objects, the outdoor scenes cover larger environments with more complex spatial structures, whereas the indoor scenes offer a more controlled layout.

Scene	Dataset	Number of images	Image dimension [pixel]
Bicycle	Mip-NeRF360	194	1237×822
Garden	Mip-NeRF360	185	1297×840
Stump	Mip-NeRF360	125	1245×825
Drjohnson	Deep Blending	263	1332×876
Room	Mip-NeRF360	311	1557×1038
Playroom	Deep Blending	225	1264×832

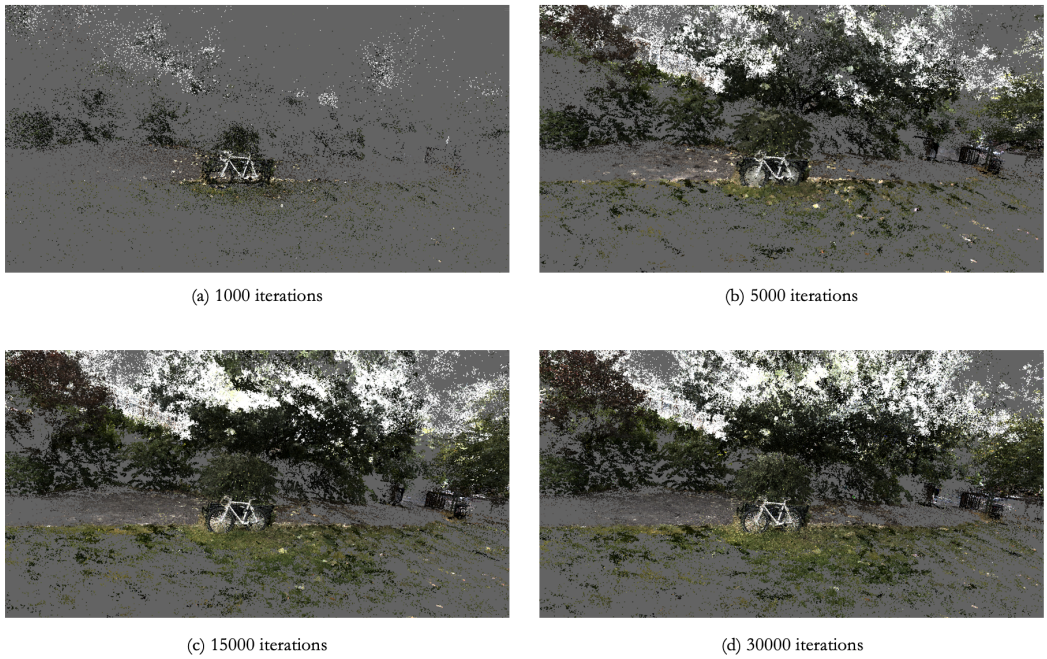
Table 4.1: Selected scenes information

4.1.2 Gaussian Splatting for scenes visualization

All scenes were trained using the Gaussian Splatting technique for 30.000 epochs. The algorithm was configured to save the scene every 1.000 iterations, resulting in 30 point clouds per scene by the end of the process. This incremental saving approach allowed us to capture the scene at various stages of the training, providing a comprehensive timeline of its reconstruction. By doing so, our dataset was effectively expanded, including intermediate stages of each model with varying levels of refinement.

By examining these different stages, we could assess how the number of splats and the level of reconstruction detail developed over time, offering insights into the algorithm's behavior at different phases of training. In particular, it allowed for the observation of how fine details and structural accuracy emerged as the training advanced, highlighting the relationship between the number of iterations and the quality of the final output.

Furthermore, this method facilitated a deeper investigation into how the algorithm responds to complex geometries and textures within the scene. By visualizing the intermediate point clouds, it became possible to identify key areas where the reconstruction accuracy significantly improved, as well as regions where progress stagnated or required additional iterations.



(a) 1000 iterations (b) 5000 iterations

(c) 15000 iterations (d) 30000 iterations

Figure 4.2: Point cloud of splats of `bicycle` scene after different iterations of 3DGS

4.2 Gaussians Density and Distribution Analysis

An initial analysis focused on the number of Gaussians and their density across different regions of the scene. The number of Gaussians used to represent the scene was monitored throughout the Gaussian Splatting training process over 30.000 epochs.

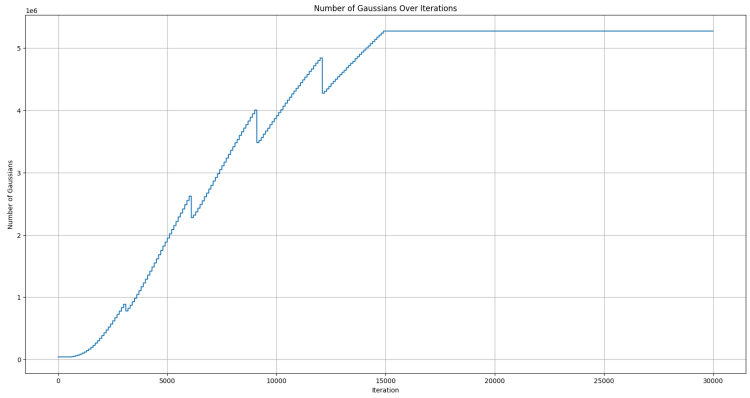


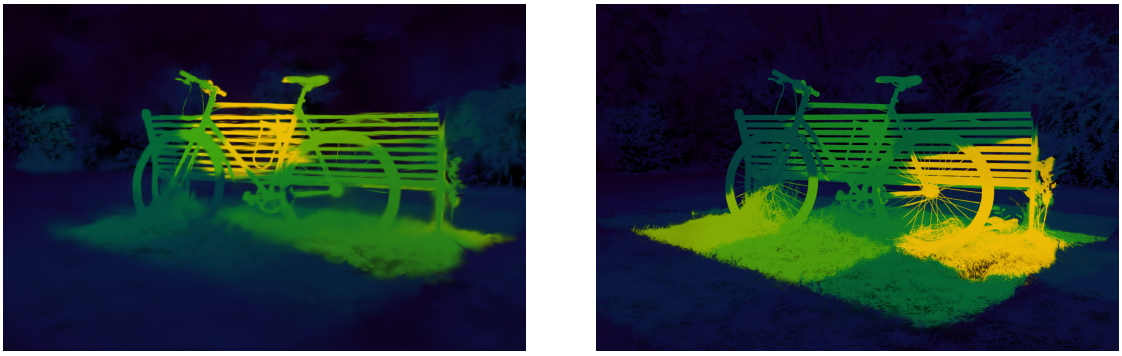
Figure 4.3: Number of Gaussians as a function of the iteration during the 3DG optimization for `bicycle` scene. As the iterations progress, the number of Gaussians increases in a stepwise manner, reaching a maximum value after approximately 15.000 iterations.

4.2.1 Voxelization

To gain deeper insight into the spatial distribution of Gaussians, a voxelization approach was adopted. The point cloud representing the scene was divided into unit-sized voxels ($1 \times 1 \times 1$), and the number of Gaussians contained within each voxel was calculated at regular intervals during training. Specifically, this analysis was conducted every 1.000 iterations of the 3DGS process, providing snapshots of the evolving Gaussian distribution throughout the optimization. This tracking allowed for a detailed observation of how the representation evolved over time, offering insights into the optimization dynamics and how Gaussians were allocated in different regions of the scene. Understanding the distribution and density of Gaussians can be useful, as it directly impacts the visual fidelity and accuracy of the rendered scenes.

Voxelization also revealed areas of the scene that became more densely populated with Gaussians as the training progressed. Regions with higher Gaussian density typically corresponded to parts of the scene requiring more detailed representation, such as those with complex geometry, fine textures, or intricate lighting. These densely populated areas indicated where the algorithm focused during optimization, highlighting the regions prioritized for higher fidelity.

To enhance visualization and understanding, the point clouds were colored based on Gaussian density, allowing for clearer identification of denser regions.



(a) 1000 iterations **(b)** 30 000 iterations

Figure 4.4: Density-based renderings of the point cloud at 1.000 and 30.000 iterations are shown. The `viridis` colormap was used to represent density, where yellow indicates higher density and blue represents lower density. It can be observed that the bicycle and the bench are the most densely represented objects in the scene, with the grass near them also exhibiting high density. Additionally, a noticeable shift in density is observed from the center of the bench toward the lower-right corner.

4.3 Quality Assessment Metrics

In image processing and computer vision, evaluating the visual quality of generated or reconstructed images is fundamental. Traditional pixel-wise comparisons often fail to reflect the way humans perceive image quality, leading to the development of more sophisticated quality assessment metrics. These metrics aim to provide objective measures that correlate with human perception, helping to bridge the gap between mathematical accuracy and perceived visual fidelity. The need for objective metrics stems from the limitations of subjective assessments, which, while more accurate, are impractical for large-scale benchmarking and automated systems.

In the scenario of Gaussian Splatting, benchmarking is often based on the use of these visual quality assessment metrics to evaluate performance. These metrics allow for consistent and reproducible evaluations of image quality, either by comparing with reference images (full-reference metrics) or assessing image quality independently (no-reference metrics). The peak signal- to-noise ratio (PSNR), structural similarity index measure (SSIM), and learned perceptual image patch similarity (LPIPS) are widely utilized in the Gaussian Splatting literature as the primary metrics for this purpose. Each of these metrics approaches the problem from a different angle: PSNR focuses on pixel-level accuracy, SSIM incorporates aspects of the human visual system by evaluating structural content, and LPIPS leverages deep neural networks to assess perceptual similarity. The

mathematical formulation of theses metrices are defined below.

4.3.1 Peak Signal to Noise Ratio

The term peak signal-to-noise ratio (PSNR) is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. Because many signals have a very wide dynamic range, (ratio between the largest and smallest possible values of a changeable quantity) the PSNR is usually expressed in terms of the logarithmic decibel scale. The mathematical definition of the PSNR is as follows:

$$\mathrm{PSNR}(I) = 10\dot{\log}\left(\frac{\max(I)^2}{\mathrm{MSE}(I)}\right) \tag{4.1}$$

where $\max(I)$ is the maximum possible pixel value in the image (255 for 8-bit integer), and $\mathrm{MSE}(I)$ is the pixel-wise mean squared error calculated on all color channels:

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}||f(i,j) - g(i,j)||^2 \tag{4.2}$$

where f represents the matrix data of the original image, g represents the matrix data of the degraded image, m represents the numbers of rows of pixels of the images and n represents the number of columns of pixels of the image.
A higher PSNR indicates that the degraded image has been reconstructed more accurately to match the original image, reflecting the effectiveness of the reconstructive algorithm. This is achieved by minimizing the mean squared error (MSE) between the images in relation to the maximum signal value of the original image.

The main limitation of this metric is that it relies strictly on numeric comparison and does not actually take into account any level of biological factors of the human vision system such as the SSIM.

4.3.2 Structural Similarity Index Measure

SSIM [51] is a full-reference quality assessment metric that measures the similarity between a reference x and the assessed y images in terms of their structural content, by comparing various statistical properties of the images, including means (μ_x and μ_y), variances (σ_x^2 and σ_y^2), and the covariance (σ_{xy}) between them. The SSIM for a single patch is given by the following formula:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.3}$$

This equation quantitatively measures how similar two images are in terms of structure and texture. The local statistics μ and σ are calculated within a 11×11 circular symmetric Gaussian weighted window, with weights w_i having a standard deviation of 1.5 and normalized to 1. Additionally, $C_1 = (K_1 L)^2$ and $C_2 = (K_2 L)^2$ are constants that prevent division by very small variances, ensuring numerical stability in the calculation. Here, L is the dynamic range of the pixels (255 for 8bit integer), $K_1 = 0.01$ and $K_2 = 0.03$ are constants chosen in [51].

SSIM incorporates aspects of the human visual system by comparing images based on structural information, focusing on luminance, contrast, and structural similarity. It models how humans perceive image quality by emphasizing the preservation of structural patterns, which are more significant to human perception than pixel-level differences. SSIM operates by comparing local patterns of pixel intensities, taking into account visual phenomena like brightness adaptation and contrast masking, which are key to how the human eye interprets visual data.

A higher SSIM value indicates a higher similarity between the two images and a better quality image. SSIM is more sensitive to subtle changes in the image than PSNR, but it is also more difficult to compute.

4.3.3 Learned Perceptual Image Patch Similarity

LPIPS [56] is a complete reference quality assessment metric that uses learned convolutional characteristics. Unlike traditional metrics like PSNR and SSIM, which rely on pixel-level differences, LPIPS focuses on higher-level image structure and visual perception. It extracts deep features from intermediate layers of pre-trained convolutional neural networks, such as VGG [42] or AlexNet [27], which have been trained on large-scale image classification tasks.

The perceptual distance between two images is computed as the weighted Euclidean distance between their deep feature representations, averaged spatially and across different network layers. The weights used to calculate these distances are learned from datasets containing human judgments on perceptual similarity, ensuring that LPIPS better aligns with human vision. This process, known as *perceptual calibration*, allows LPIPS to adapt to subtleties in human perception, improving its ability to predict human judgments of image similarity.

The score is given by a weighted pixel-wise MSE of feature maps over multiple layers.

$$\text{LPIPS}(x, y) = \sum_{l=1}^{L} \frac{1}{H_l W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} ||w_l(x_{lhw} - y_{lhw})||_2^2 \tag{4.4}$$

where x_{lhw}, y_{lhw} represent the features of the reference and assessed images at the pixel at

coordinates (w, h), and layer l. W_l and H_l are the width and height of feature map at layer l.

LPIPS is the most recent metric and the most sensitive to subtle changes in the image. It is able to capture the subtle differences between images that are not captured by traditional metrics such as PSNR and SSIM. However, it is also the most computationally expensive metric.

5

Proposed method

This project aims to develop an optimized method for real-time rendering of 3D scenes using Gaussian splatting. The goal is to create an adaptive rendering process that maintains high-quality visualization across the entire scene while optimizing resource allocation.

Our approach relies on the integration of semantic segmentation, which allows us to distinguish and prioritize objects based on their relevance ad complexity. By identifying which areas of the scene require more detailed rendering and which can be simplified without noticeable quality loss, we can selectively optimize different regions. Additionally, the distance of each object from the viewer serves as a discriminating factor, further guiding the optimization process. The proposed pipeline combines image-based semantic segmentation, which helps categorize objects; 3D reconstruction, which maps these segmented objects into the point cloud; and curve fitting, which predicts the optimal iteration for halting Gaussian splatting while achieving the desired SSIM. The combination of these methods enables the generation of high-quality, dynamic visualizations that adapt to the specific requirements of each scene, thus ensuring visual fidelity without unnecessary computational overhead.

5.1 Semantic Segmentation

Semantic segmentation is a fundamental task in computer vision that involves labeling each pixel in an image with a predefined class, such as "sky," "road," or "car." The goal is to classify every pixel, allowing for a detailed understanding of the scene at the pixel level. Formally, given an input image I of size $H \times W$, a deep learning model, typically based on Convolutional Neural Networks (CNNs), is trained to map the image to a function f that assigns each pixel a probability distribution over the possible semantic classes. The final label for each pixel is determined by selecting the class with the highest probability. This process can be seen as clustering the image into regions where

pixels within the same region share the same semantic label. Mathematically, for a set of N pixels $X = \{x_1, x_2, ..., x_N\}$, each pixel $x_i \in \mathbb{R}^2$ is assigned one of k labels from a set $Y = \{y_1, y_2, ..., y_k\}$.

Semantic segmentation can also be extended beyond two-dimensional images to three-dimensional data, where the task is to label 3D points instead of pixels.

In this thesis, semantic segmentation is used as an alternative scene analysis method to the voxelization described in Sec. 4.2.1. Semantic segmentation enables object-level reasoning, allowing us to assess how 3DGS optimization performs on each object. The goal is to create scene visualizations with different levels of detail that maintain overall visual quality while allocating resources more efficiently. To achieve this, semantic segmentation is applied to the input images (Sec. 5.1.1) used in the scene reconstruction algorithm to generate 2D masks, which are then projected onto the 3D point cloud using majority voting to produce a 3D scene segmentation (Sec. 5.1.2).

5.1.1 2D Semantic Segmentation

Semantic segmentation was performed on all input images of the following scenes: `bicycle`, `garden`, `stump`, `room`, `playroom`, and `drjohnson`. This process produced a full set of 2D masks, with the goal of generating a mask for each original image, where semantic labels are assigned to the pixels. To achieve this, two algorithms, SAM [26] and DeepLab2 [52], were tested on the `bicycle` dataset. Based on the results, DeepLab2 was chosen for the final implementation.

5.1.1.1 **SAM**

The Segment Anything Model (SAM) [26] is a general-purpose segmentation model developed by Meta AI, designed to segment objects within an image without requiring fine-tuning for specific tasks or datasets. SAM is a prompt-based model, meaning it can accept different types of inputs — such as points, bounding boxes, or text descriptions — to guide the segmentation process. This flexibility makes it highly adaptable for various tasks, ranging from precise object segmentation to broad scene parsing. SAM leverages a large pre-trained model based on a Vision Transformer (ViT), enabling generalization across diverse domains and datasets, and handling a wide variety of object categories.

In our experiments, SAM was evaluated on the `bicycle` dataset to assess its ability to generate accurate 2D semantic masks. Using SAM's automatic mask generation without prompts, more than 90 binary masks were produced per image, requiring manual selection of relevant masks. Objects such as the bicycle, bench, road, grass, and trees were selected. While SAM generally performed well, there were instances where it struggled to distinguish between the bicycle and bench, particularly when the bench was in the foreground and the bicycle was behind it, leading to their fusion into a single mask.

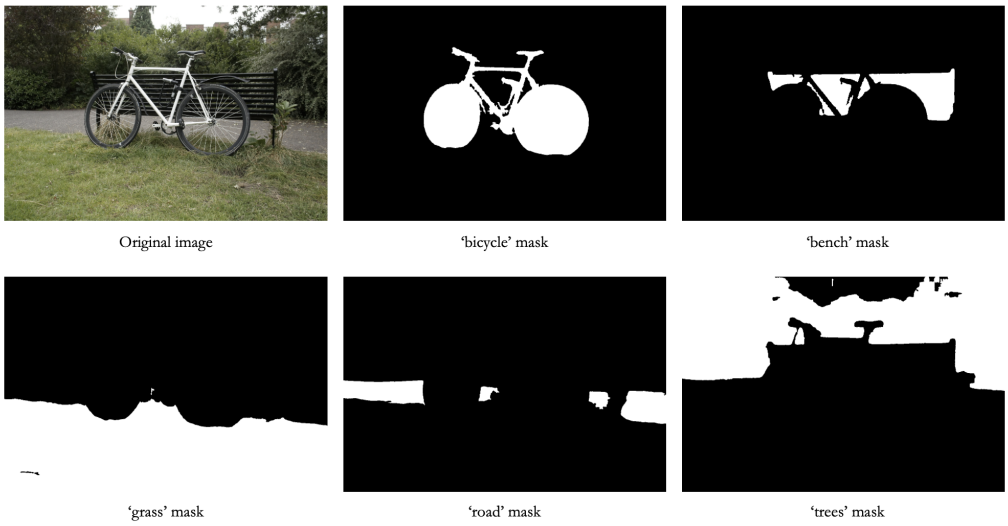


Figure 5.1: Selected binary masks (white identify the object) generated by SAM for the first image of `bicycle` dataset

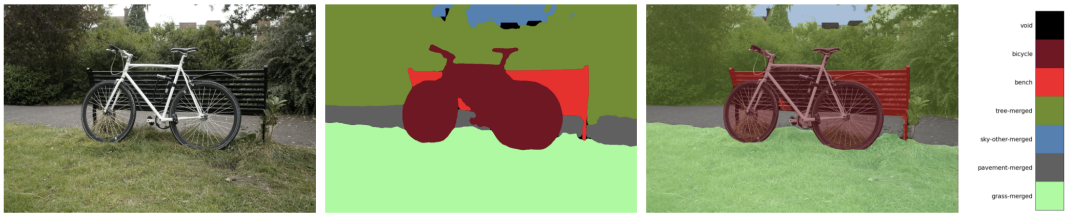
Masks representing small objects, such as stones, or individual parts of larger objects, like bicycle wheels, were typically discarded.

Although SAM demonstrated robustness and flexibility, the large number of generated masks and the complexity of integrating prompts for large-scale processing — particularly in a context where automation is key — led us to explore other models. Specifically, we chose DeepLab2, which offered a more streamlined and consistent approach for generating 2D masks automatically.

5.1.1.2 DeepLab2

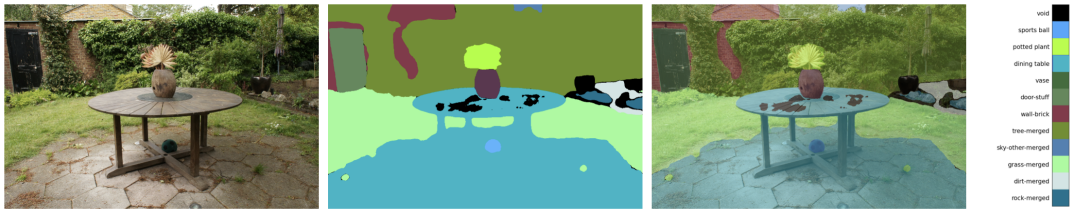
DeepLab [9] is a series of deep learning architectures designed for semantic segmentation, leveraging atrous convolution (also known as dilated convolution) to enhance performance. In the original DeepLab model [9], atrous convolution is employed to control the resolution of features computed by CNN backbones. This approach allows for an expanded receptive field without increasing the number of parameters, making the model more efficient. Later versions of DeepLab [8] introduce the ASPP (Atrous Spatial Pyramid Pooling) module, which enhances the model's ability to capture multi-scale information by applying parallel atrous convolutions with different sampling rates.

DeepLab2 [52] is a TensorFlow library for deep labeling, aiming to provide a state-of-the-art and easy-to-use TensorFlow codebase for general dense pixel prediction problems in computer vision. It includes various DeepLab model variants, pretrained checkpoints, and code for model training and evaluation. In this thesis, DeepLab2 was utilized with checkpoints pretrained on the COCO dataset [30] to segment all images from the selected scenes automatically.



Original image Semantic mask Semantic overlay

(a) `bicycle`



Original image Semantic mask Semantic overlay

(b) `garden`

Original image Semantic mask Semantic overlay

(c) `room`

Figure 5.2: Mask generated by DeepLab2 for the first image of (a) `bicycle` scene, (b) `garden` scene and (c) `room` scene

5.1.2 3D Semantic Segmentation

The 2D semantic segmentation results are projected into 3D space to segment the point cloud. This process becomes important when we want to synthesize novel views of the scene. To render a new view, semantic information is required, but since the image is not available in advance, having the semantic information embedded in the point cloud allows for the creation of semantic masks by projecting this data onto the 2D plane. Additionally, 3D semantic information is important for the final task of halting training at different iterations for different objects, enabling more efficient resource allocation during the reconstruction process.

To project 2D semantic masks onto the 3D point cloud and obtain a segmented point cloud, the process begins by associating each 2D image with its corresponding camera parameters, including pose (rotation and translation matrices) and intrinsics (focal length and optical center). These parameters, obtained from COLMAP reconstruction [39], are used to project each 3D point onto the 2D image plane. For each point in the point cloud, the projection determines its corresponding pixel in the 2D image, allowing it to inherit the semantic label assigned to that pixel. Since the same 3D point may be visible in multiple images from different viewpoints, it can receive different semantic labels depending on the image from which it is projected. To resolve this, majority voting is applied: the semantic label that appears most frequently across the projections is assigned to the 3D point.

5.2 Unity setting

Unity 3D [48] is a versatile real-time development platform, known for its ability to create immersive 2D, 3D, AR and VR experiences across a wide range of industries. Initially developed for game design, Unity has expanded its reach to fields offering powerful tools for real-time visualization, interactive simulations, and more. Its user-friendly interface, combined with extensive scripting capabilities, allows developers to efficiently prototype and build high-quality interactive content.

In this work, Unity was utilized as a tool for visualization and extended reality simulations. The scenes were imported into the platform and visualized using the "Gaussian Splatting for Unity" tool [47]. Unity was also employed to recreate the images used in model reconstruction by rendering scenes saved at different iterations, providing a means to evaluate their accuracy.

5.2.1 Gaussian Splatting Tool

The Gaussian Splatting tool for Unity [47] is an experimental implementation designed to visualize 3D scenes using 3D Gaussian splatting. It provides an efficient approach

for rendering large Gaussian splat datasets within Unity, leveraging compression and optimization techniques for improved performance.

The tool allows users to create customized GaussianSplat assets from Gaussian Splat PLY files, which are point clouds of trained splats. Once the asset is generated, it can be assigned to a `GaussianSplatRenderer` script attached to a game object. This script provides control for debugging and visualizing the data. The tool takes into account game object transformation matrices, which is useful for models that may require adjustments such as rotation and mirroring. For instance, models are often rotated by 180 degrees around the X-axis and mirrored around the Z-axis.
The integration of this tool into Unity provides a robust, high-performance framework for rendering with 3DGS large point cloud models in Unity, making it suitable for both real-time applications and static scenes.

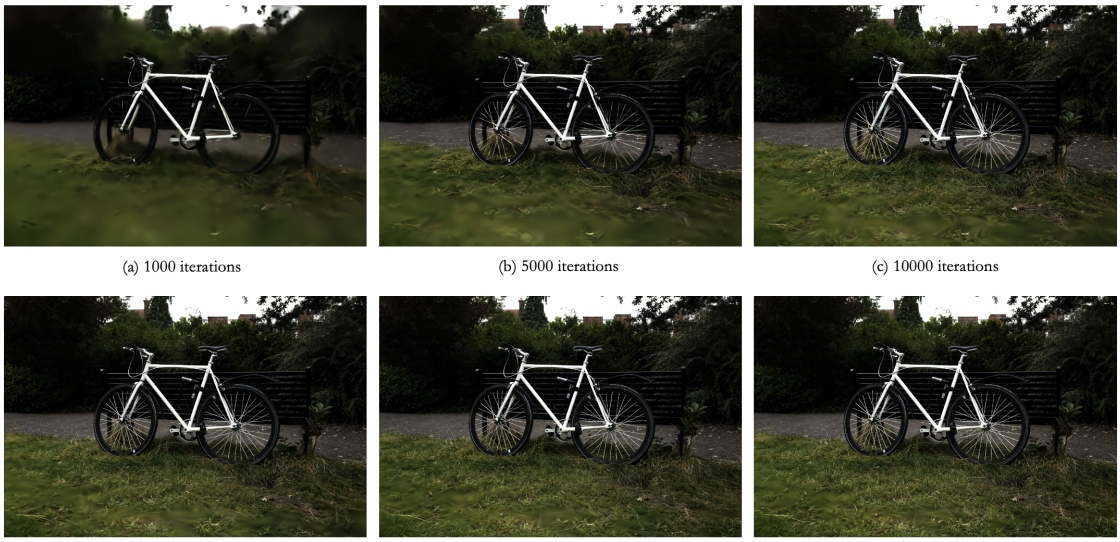
5.2.1.1 Camorph

Camorph [4] is a Python-based toolkit designed to convert between various camera parameter representations, with functionality available both as a library and through a command-line interface. It offers key features such as fast and intuitive conversion between different camera parameter conventions, seamless management of essential properties across multiple formats, and automated calculation of intrinsic camera parameter representations.

In this thesis, Camorph is used to convert the camera parameters generated by COLMAP during scene reconstruction into Unity's format. This ensures that the camera viewpoints in Unity accurately match the real-world perspectives at the time the photographs were taken.

5.2.2 Rendering Captures

The goal is to recreate the original photographs of the scenes by rendering the models saved at different iterations, in order to compute SSIM, PSNR, and LPIPS between these images and the ground truth. To achieve this, a Unity scene was created for each model in our dataset, importing the cameras obtained from the conversion with Camorph. These cameras are correctly set with the intrinsic and extrinsic parameters derived from the scene reconstruction, ensuring accurate camera calibration. All point clouds saved during the optimization of 3D Gaussian Splatting were imported as GaussianSplat assets. For each scene, at each iteration where the model was saved, a script ensures that, upon launching the application, each camera captures an image of the scene. These images are saved and later used to compute the relevant metrics. The background of the Unity cameras is set to black, so in cases where there are "holes" in the model, the black background is rendered. The resolution of the captured images matches that of the original images, as specified in Tab. 4.1.



(a) 1000 iterations (b) 5000 iterations (c) 10000 iterations

(d) 15000 iterations (e) 20000 iterations (f) 30000 iterations

Figure 5.3: Rendering of `bicycle` scene in Unity captured by the camera corresponding to the first original image

5.3 Quality Metrics Evaluation

Visual quality metrics presented in Sec. 4.3 are computed on the captured images, taking as ground truth the original images of the scenes. Specifically, these metrics are computed both for the entire images and for semantic-masked images. Before proceeding with the computation, histogram matching is applied.

5.3.1 Histogram Matching

Histogram matching is a technique used to adjust the intensity distribution of an image so that its histogram matches that of a reference image. This process aligns the pixel value distribution between two images, making them more visually similar.

In our case, two specific functions were applied to the images captured in Unity for this purpose. The first function focuses on matching the mean and standard deviation of the input image I_n where n is the iteration at which the 3DGS was stopped, to those of the reference image I_{ref} for each color channel (RGB). It adjusts the pixel values of image I_n by shifting and scaling them based on the statistical properties of image I_{ref}. This color transfer ensures that the overall appearance of the two images is more consistent, which is important for reducing visual differences. The second function directly performs histogram matching. It first calculates the histograms and Cumulative Distribution Functions (CDFs) for both images, then remaps the pixel values of image A to align with the intensity distribution of image B.

By applying these two methods, the rendered images are transformed to closely match the original ones, minimizing discrepancies that could otherwise impact the accuracy of the SSIM, PSNR, and LPIPS metrics.

Figure 5.4: Comparison between different image processing techniques applied to a bicycle scene. On the far left is the Ground Truth, representing the original reference image. Next is the simulated rendering of the scene in Unity, where noticeable lighting and color differences can be observed. The third image shows STD Matching, where the Unity capture has been adjusted using standard deviation matching to better balance brightness and contrast, bringing it closer to the ground truth. Finally, the last image illustrates Histogram Matching, where the Unity image has undergone histogram matching to modify its color distribution and further resemble the ground truth.

5.3.2 Image-based Quality Metric

PSNR, SSIM, and LPIPS are initially computed on the entire images by comparing each image at different iterations with its corresponding original from the dataset. As expected, progressing through iterations results in higher-quality images. However, these metrics represent average values across the entire image. To allocate resources more efficiently, it is important to understand how image quality varies across specific regions. To achieve this, a preliminary approach involves dividing the images into smaller square regions and computing the metrics for each section separately. This allows us to assess localized changes in quality, highlighting areas with significant improvements as well as those that exhibit poorer performance.

5.3.3 Semantic-based Quality Metric

Our goal is to achieve a label-based optimization, leveraging scene semantics to identify objects and evaluate the amount of resources each object requires based on its complexity and distance. Since we are working at the object level, it makes sense to compute SSIM and PSNR for individual objects to assess how each one is affected by the progression of the Gaussian Splatting training. Therefore, the images were masked using the semantic masks obtained earlier (Sec. 5.1.1.2), and PSNR and SSIM were computed for each mask, resulting in label-based metrics. In this step, LPIPS was not calculated, as it is not straightforward to apply this metric to part of images of complex shapes.

5.3.3.1 **PSNR**

PSNR was the first metric analyzed, as 3DGS also provides a PSNR measure of the model during training. However, when examining PSNR for individual objects, we observed that this metric does not fully capture the quality of the analyzed scene segments. For instance, when examining images of the `bicycle` scene objects at different iterations, it is evident that in the first 5000 iterations, the grass appears very blurred, while it improves at higher iterations. Yet, the PSNR curve remains relatively constant, suggesting that quality does not increase significantly with more iterations. For this reason, we decided to focus more on SSIM, which provided more representative results.

5.3.3.2 **SSIM**

SSIM, on the other hand, proved to be a metric that effectively represents the evolution of object quality as the iterations progress. For each scene, the SSIM values calculated for the same label across different images were compared to evaluate the quality trend for that specific object. It was observed that the relationship between 3DGS iterations and SSIM follows a logarithmic pattern. For each label in each scene, we identified the curve with the highest average SSIM, as well as the curve representing the overall average SSIM.

The relationship between SSIM and object distance was also investigated to understand how the distance of an object from the camera affects its visual quality and how this changes over different iterations. The minimum distance was used for this analysis, calculated as the closest point of the 3D point cloud projected onto the semantic mask. It was found that SSIM and distance do not have a linear relationship. The SSIM curve, based on minimum distance, initially tends to decrease and then, after a certain threshold, begins to increase again, although some noise is present in the data. This trend is more noticeable for individual objects, whereas for classes that cover larger areas, like trees or grass, the initial decrease is less evident, and the subsequent growth beyond a certain distance is more linear. The relationship found is reasonable: when an object is very close, all its details are visible, and the rendering quality must be higher. As the distance increases up to a certain point, the object's resolution remains high, but not all details are well rendered, even though they are still distinguishable. Beyond this threshold, the resolution of the object drops, and details become less perceivable, making the visual quality appear to improve with increasing distance.

The semantic-based iteration-SSIM and distance-SSIM curves form the starting point for our optimization method.

5.4 SSIM fitting

Curve fitting refers to the process of determining a mathematical model that has the best fit to a series of data. In this process, we are provided with raw data and a function with unknown coefficients is defined. The goal is to find values for the coefficients such that the function matches the raw data as well as possible. The simplest case of curve fitting is linear regression, where data is fit to a straight line.

Our objective is to develop a model that determines, for each object, the iteration at which the optimization of Gaussian splatting can be stopped while achieving an SSIM equal to or greater than a specified threshold. To this end, our approach involves fitting the SSIM curves as a function of iteration and minimum distance. This allows us to optimize scene visualization by making the best use of available resources. Consequently, we need to define an objective function that relates SSIM, iteration, and distance through unknown coefficients. By fitting the SSIM curves, we aim to estimate these coefficient values, thereby defining a mathematical function capable of predicting, for each class within the scene, the iteration at which 3D Gaussian Splatting can be halted to meet the target SSIM.

This process is applied to each scene and for every class recognized within the scenes, using the same objective function. However, each class has its own set of coefficients.

5.5 Evaluation methodologies

To evaluate the performance of our models obtain from fitting and ensure that the estimated iterations lead to high-quality rendering, two types of evaluation tests are performed: the cross-view test and the cross-model test. These tests are designed to measure both the adaptability and robustness of the model across different views and scenes, ensuring that our approach not only optimizes rendering within a single scene but also generalizes well to new scenarios.

5.5.1 Cross-view test: Novel View Synthesis

The cross-view test focuses on generating a novel view of the scene by utilizing the model that has been developed specifically for the classes within that scene. This test allows us to assess how well the model performs in synthesizing unseen viewpoints while maintaining the same quality thresholds defined during the optimization process.

By leveraging the 3D segmentation of the point cloud, we can identify the points that belong to each class within the scene. This classification enables us to segment the scene based on class labels, making it possible to extract semantic masks for the novel view. The scene is then reconstructed by combining objects and features at different iterations, as determined by their respective minimum distances. This process allows for more efficient allocation of computational resources by focusing on key areas that require more iterations for high-quality rendering.

During the rendering process, each class is projected onto the screen, and for each class, we calculate the minimum distance from the viewpoint to determine the necessary iteration. Given a predefined target SSIM, our model calculates the optimal iteration needed for each class to meet or exceed the desired quality. This approach helps reduce the number of points required to represent the scene, optimizing performance while still preserving a high level of visual fidelity.

Once the new view has been synthesized, the SSIM is evaluated on two levels: first, for the entire view to check overall image quality, and second, for each semantic mask to ensure the model accurately predicts quality for individual objects or regions.

5.5.2 Cross-model test

The cross-model test takes the evaluation one step further by testing the portability and generalization of the model across different scenes. In this test, the model developed for a specific class in one scene is applied to the same class in another scene. This allows us to examine whether the learned parameters and iteration values for a class are general enough to be transferred across different scenes without sacrificing quality. For instance, if an object like a chair appears in two different scenes, the cross-model test checks whether the model trained on the chair in scene 1 can be applied to the chair in scene 2. The goal is to verify if the SSIM curve for the class remains consistent across scenes, and whether the model can predict the correct iteration to reach the desired SSIM for the object in the new context.

Our selection of scenes has been designed to facilitate this type of testing, as they were chosen specifically to contain common objects that can be analyzed across multiple environments. By comparing the SSIM values for the same class in both scenes, we validate whether the model provides consistent iterations for achieving the target SSIM.

6

Results

The following sections present the results obtained by applying the methods described in the previous chapter. For clarity, all results reported here refer to the `bicycle` scene.

6.1 Semantic Segmention Results

The semantic segmentation of the 194 images from the `bicycle` scene resulted in the identification of 17 distinct semantic classes. These classes, along with their corresponding frequencies across the images, are listed in Tab. 6.1. This table provides the distribution of data available for each semantic class. A higher occurrence frequency for a class indicates a richer dataset for that particular element, allowing for more robust and reliable analysis during the rendering process. Conversely, classes with fewer occurrences present a challenge for accurate curve fitting and prediction, as they are represented by less data, leading to more sparse sampling. To maintain statistical reliability, only semantic classes that appear at least 10 times will be considered for curve fitting and further analysis. In addition to the frequency of occurrence, the table also reports the presence of these semantic classes in other scenes from our dataset. This information allows for cross-model testing and transferability of the learned models.

For the `bicycle` scene in particular, we observe that the most frequently occurring elements, with over 50% presence in the images, include the bench, bicycle, grass, pavement, and trees.

Semantic label	Frequency (over 194)	Other scenes with same label
Bench	178	`garden`
Bicycle	107	
Building-other-merged	17	
Car	22	`room`, `playroom`
Chair	1	`garden`, `room`, `playroom`, `drjohnson`
Dirt-merged	28	`garden`, `stump`
Fence	6	
Grass-merged	166	`garden`, `stump`
House	35	`garden`
Pavement-merged	101	`garden`, `playroom`, `drjohnson`
Road	13	
Rock-merged	2	`garden`, `stump`
Roof	10	`garden`
Sky-other-merged	75	`garden`, `stump`
tree-merged	172	`garden`, `stump`, `room`
Wall-brick	16	`garden`
Window-other	1	`garden`, `room`, `drjohnson`

Table 6.1: Identified semantic classes and corresponding occurrences for `bicycle` scene

6.2 Quality Metrics Results

6.2.1 Image-based Quality Metric Results

The block-wise analysis of the images provided initial insights into which metrics are most significant for our method and validated our hypothesis that different areas of the image respond differently to 3D Gaussian Splatting (3DGS) optimization. This suggests that there is potential for targeted optimization, where specific regions can be processed differently based on their characteristics. In Figure 6.1, the PSNR, SSIM, and LPIPS results are shown for the first image of the `bicycle` scene after 1000, 15000, and 30000 iterations of training.

From the analysis, it is evident that PSNR does not show substantial differences between the blocks, maintaining fairly uniform values across the image. In contrast, SSIM and LPIPS highlight a central area that achieves high visual quality early in the optimization process, whereas the bottom row of blocks struggles to reach comparable results. Looking at the results over time, there is a clear improvement in quality during the first 15000 iterations, while the difference between 15000 and 30000 iterations is less pronounced. This indicates that certain portions of the image may not require the full 30000 iterations to meet the desired visual quality, suggesting that a similar or comparable quality can be achieved with fewer iterations and, consequently, fewer points.

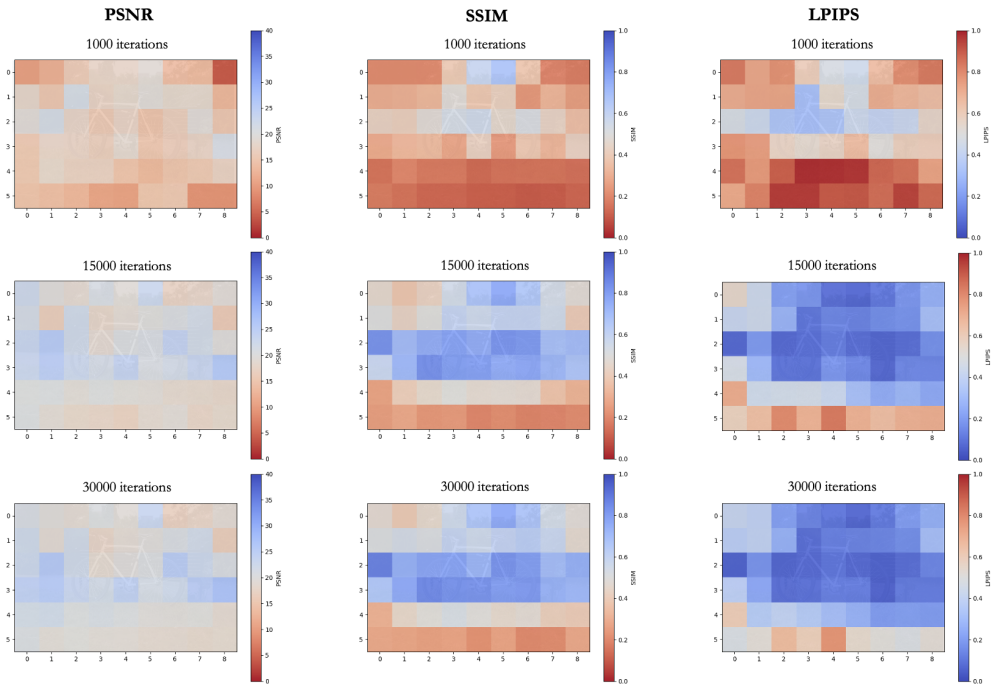


Figure 6.1: Comparison of image quality metrics across iterations for different regions of the scene. The left columns represents PSNR, the central column shows SSIM, and the right column displays LPIPS, while rows correspond to results at 1000, 15000, and 30000 iterations. Higher PSNR and SSIM values indicate better quality, while lower LPIPS values represent improved perceptual similarity. The heatmaps show how these metrics vary spatially across different patches of the scene as the number of iterations increases.

6.2.2 Semantic-based Quality Metric Results

Since we aim to develop an object-based approach, we shifted our analysis from image blocks to semantic classes. This allowed us to compute SSIM and PSNR metrics for each semantic class across all images and for each saved iteration during the training process. The results revealed that each object exhibits a slightly different performance curve, suggesting that every object could potentially follow its own mathematical model depending on its complexity and texture. Additionally, the position of the object within the image plays a crucial role, as proximity to or distance from the viewer affects the level of detail captured. Objects closer to the camera may reveal finer details, while those farther away may exhibit fewer details, influencing the metrics. This indicates the need for an adaptive model that considers not only the object's characteristics but also its spatial context within the scene to optimize rendering effectively.

We obtained two types of results: view-dependent and label-dependent. The view-dependent results present performance curves for the different semantic classes recognized within a specific view, capturing how each class performs as the view evolves over iterations. On the other hand, the label-dependent results provide aggregated insights

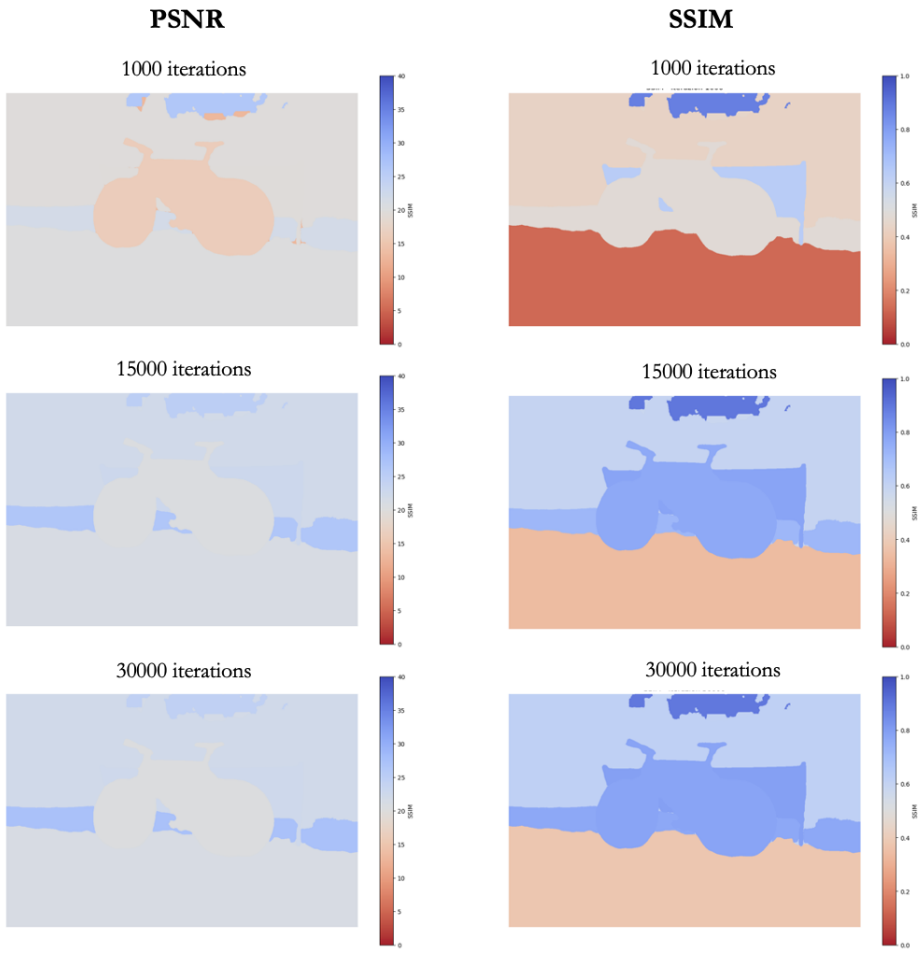


Figure 6.2: The visualization of PSNR (left column) and SSIM (right column) metrics at various stages of iteration—specifically at 1000, 15000, and 30000—is presented through heatmaps. These heatmaps illustrate the spatial distribution of the metrics across the semantic classes, utilizing a reversed `coolwarm` colormap, where blue indicates good visual quality and red signifies poor quality. PSNR values exhibit similar values across different classes. In contrast, the SSIM results demonstrate varying performance among different objects, highlighting the impact of semantic complexity on perceived visual quality.

by calculating the mean and maximum SSIM values for each semantic class across all views over the epochs. This dual approach allows us to analyze how individual objects perform in specific viewpoints, while also providing a broader understanding of each class's overall quality across the entire scene.

6.2.2.1 View Dependent Results

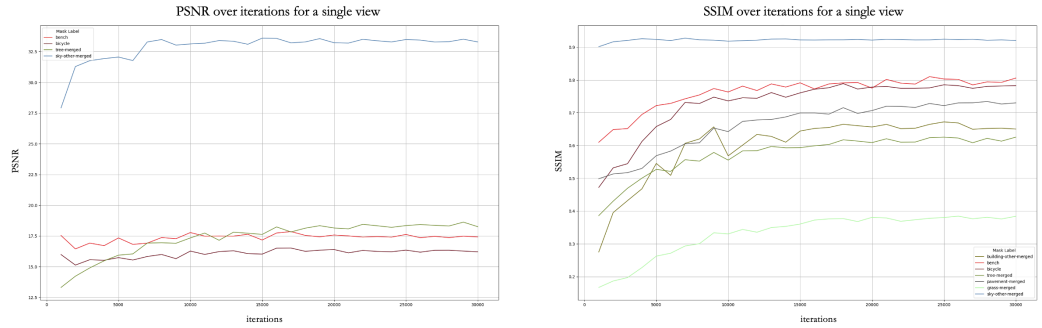
View-dependent results focus on the performance of semantic classes within individual views, highlighting how the visual quality evolves over iterations for each class as

the rendering progresses. By analyzing these results, we can observe the behavior of different objects in the scene under varying perspectives and levels of detail, with the distance of each object fixed within a single scene. This analysis allows us to identify which elements benefit most from increased iterations and how object complexity and positioning influence the overall visual quality.

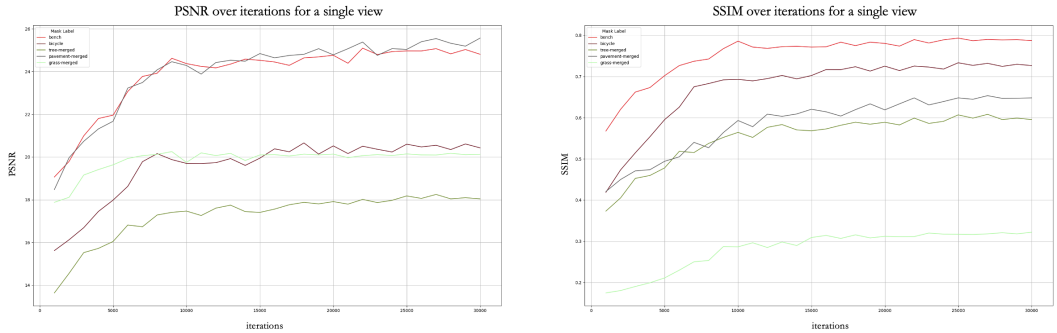
In Fig. 6.3, the results for four different views are presented. Although objects are positioned differently from the camera in each view, their SSIM curves exhibit similar trends, albeit with slight variations in absolute values. As anticipated, PSNR has proven to be a less descriptive metric for our study. For instance, in views 2 and 3, the object curves show distinguishable, steady improvements over iterations, whereas in views 1 and 4, PSNR values remain relatively constant throughout optimization, despite clear visual differences in the rendered images. Focusing on SSIM, it is evident that SSIM exhibits a logarithmic relationship with the number of iterations. Specifically, across all plots, the bicycle class shows a rapid and pronounced increase in SSIM during the first 10.000 iterations, followed by a slower, yet steady rise. This behavior is likely due to the complexity and detailed nature of the bicycle, which requires more precise rendering early in the optimization process. Conversely, the grass class consistently shows lower performance compared to other classes, reflecting its simpler visual features and lesser need for detail in the optimization process.

The trend of the SSIM is studied as a function of the distance of semantic classes from the camera in the scene at fixed iteration. Distance is closely related to the view of the scene, and each photo represents a sample for our analysis. The SSIM trend based on distance, specifically the minimum distance, was plotted for each object. In this case, the data were filtered using a median filter with window size of 9 to remove outliers likely caused by particularly unfavorable viewpoints, which would skew the trend. The Gaussian filter with sigma set to 2.5 is tested.

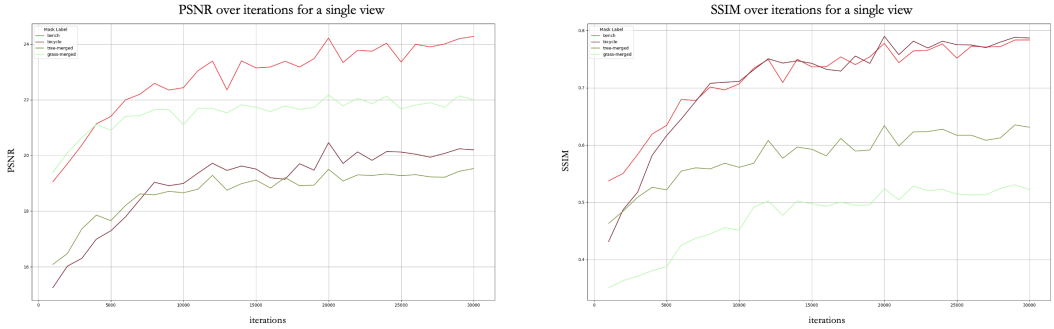
As observed, SSIM starts high for very small distances, with some variability due to the point of view. The trend initially decreases, but after a certain threshold distance, the SSIM value begins to increase. This behavior is explained by the fact that after a specific threshold, the object's resolution decreases, and its details become less distinguishable. At this point, the object is distant enough to be perceived as high-quality even without all its details. The 2D display on a screen is a pixel-quantized representation, and as the object's distance from the camera increases, the number of pixels representing the object decreases. After a certain distance, some details are lost due to quantization because the number of pixels is insufficient to represent the finer details. In these cases, the object appears small on the screen, and the loss of detail is not perceived as a loss of quality by the user.



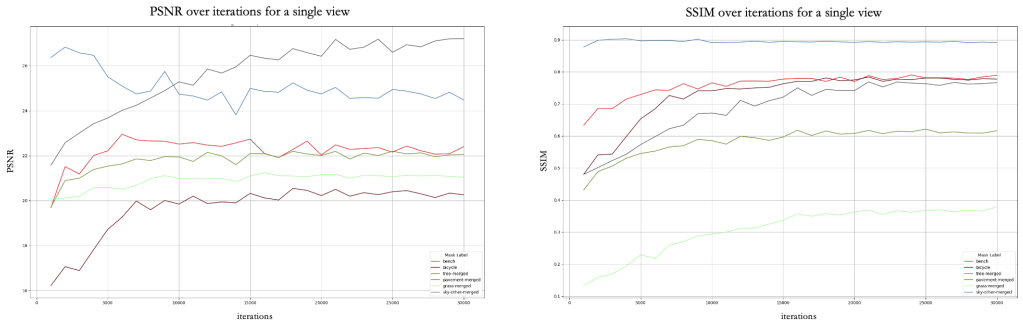
(a) View 1



(b) View 2



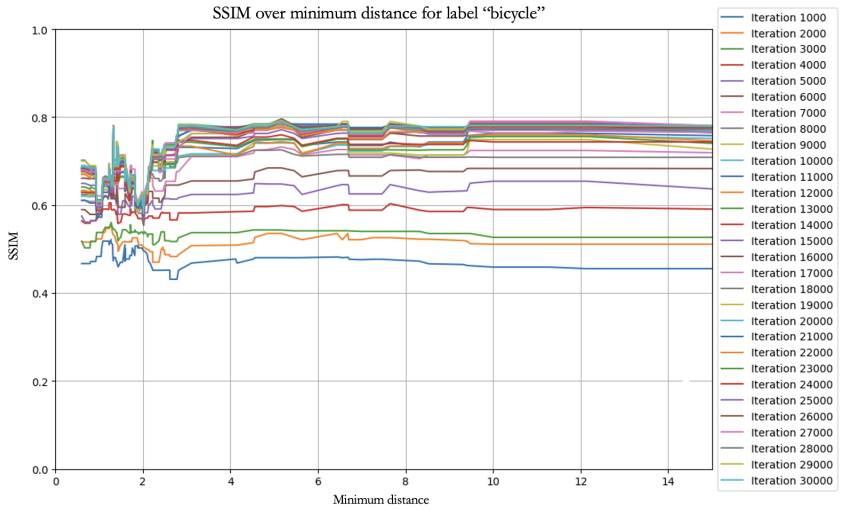
(c) View 3



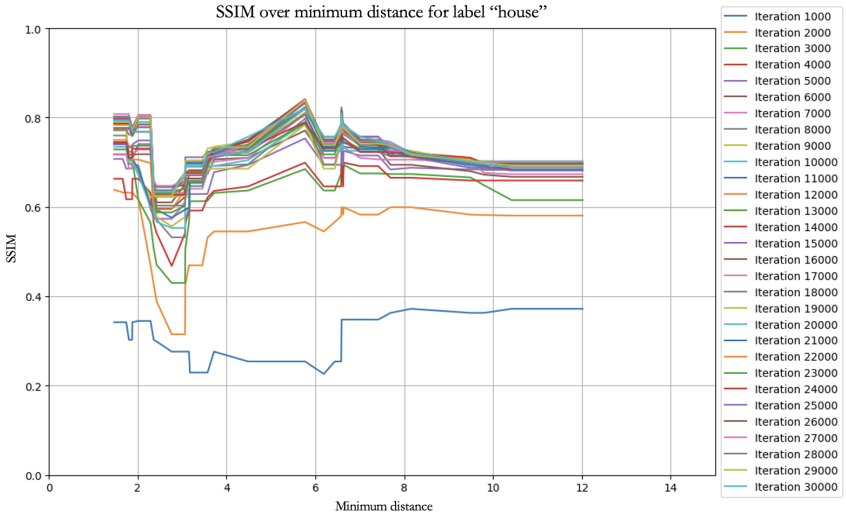
(d) View 4

Figure 6.3: PSNR and SSIM metrics over iterations for different segmentation labels in a single view. The curves represent the performance of each mask label.

The relationship between minimum and maximum distances of points for each semantic class was also analyzed, with SSIM on the x-axis, to evaluate the range of distances corresponding to a given SSIM value. It can be observed that there is much more variance in the maximum distances than in the minimum distances. For example, in the case of grass, it makes sense that the minimum distance is very low, as grass often appears in the foreground. However, the maximum distance is more strongly influenced by the camera framing.



(a) Bicycle



(b) House

Figure 6.4: SSIM over minimum distance for different labels

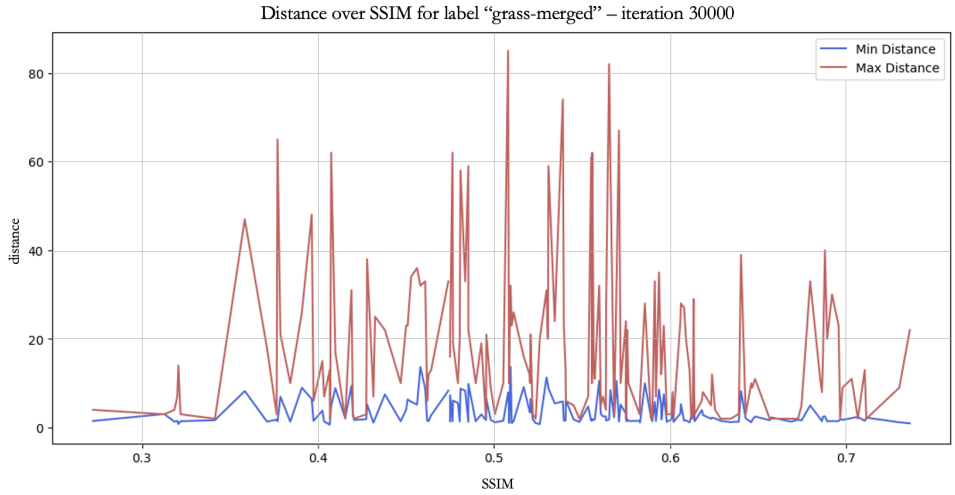


Figure 6.5: Minimum and maximum distances over SSIM values for the semantic class "grass-merged" at iteration 30,000. The blue line represents the minimum distance, while the red line shows the maximum distance as SSIM changes. The graph highlights the variation in spatial distance metrics as SSIM improves, showing the different ranges of minimum and maximum distances observed across the scene.

6.2.2.2 Label Dependent Results

The SSIM values for the labels were evaluated without correlating them to a specific viewpoint. To obtain a more global assessment, the average SSIM value and the maximum value obtained at each iteration were calculated, considering all the curves derived from various views for each specific label. Table 6.2 presents the maximum and average SSIM values for the five classes with highest occurency rate at every 5000 iterations.

	Bench		Bicycle		Grass		Pavement		Trees	
Iter	$\overline{\text{SSIM}}$	SSIM_{max}	$\overline{\text{SSIM}}$	SSIM_{max}	$\overline{\text{SSIM}}$	SSIM_{max}	$\overline{\text{SSIM}}$	SSIM_{max}	$\overline{\text{SSIM}}$	SSIM_{max}
1000	0.504	0.678	0.511	0.690	0.357	0.637	0.512	0.721	0.414	0.648
5000	0.598	0.765	0.602	0.757	0.438	0.677	0.585	0.740	0.540	0.700
10000	0.635	0.798	0.658	0.786	0.491	0.707	0.625	0.785	0.574	0.706
15000	0.659	0.823	0.674	0.814	0.521	0.719	0.642	0.802	0.594	0.720
20000	0.671	0.846	0.687	0.812	0.532	0.725	0.645	0.829	0.604	0.720
25000	0.679	0.847	0.688	0.817	0.535	0.731	0.650	0827	0.610	0.727
30000	0.682	0.855	0.688	0.817	0.537	0.736	0.651	0.830	0.613	0.735

Table 6.2: Summary table of mean SSIM ($\overline{\text{SSIM}}$) and maximum SSIM (SSIM_{max}) values at various iterations

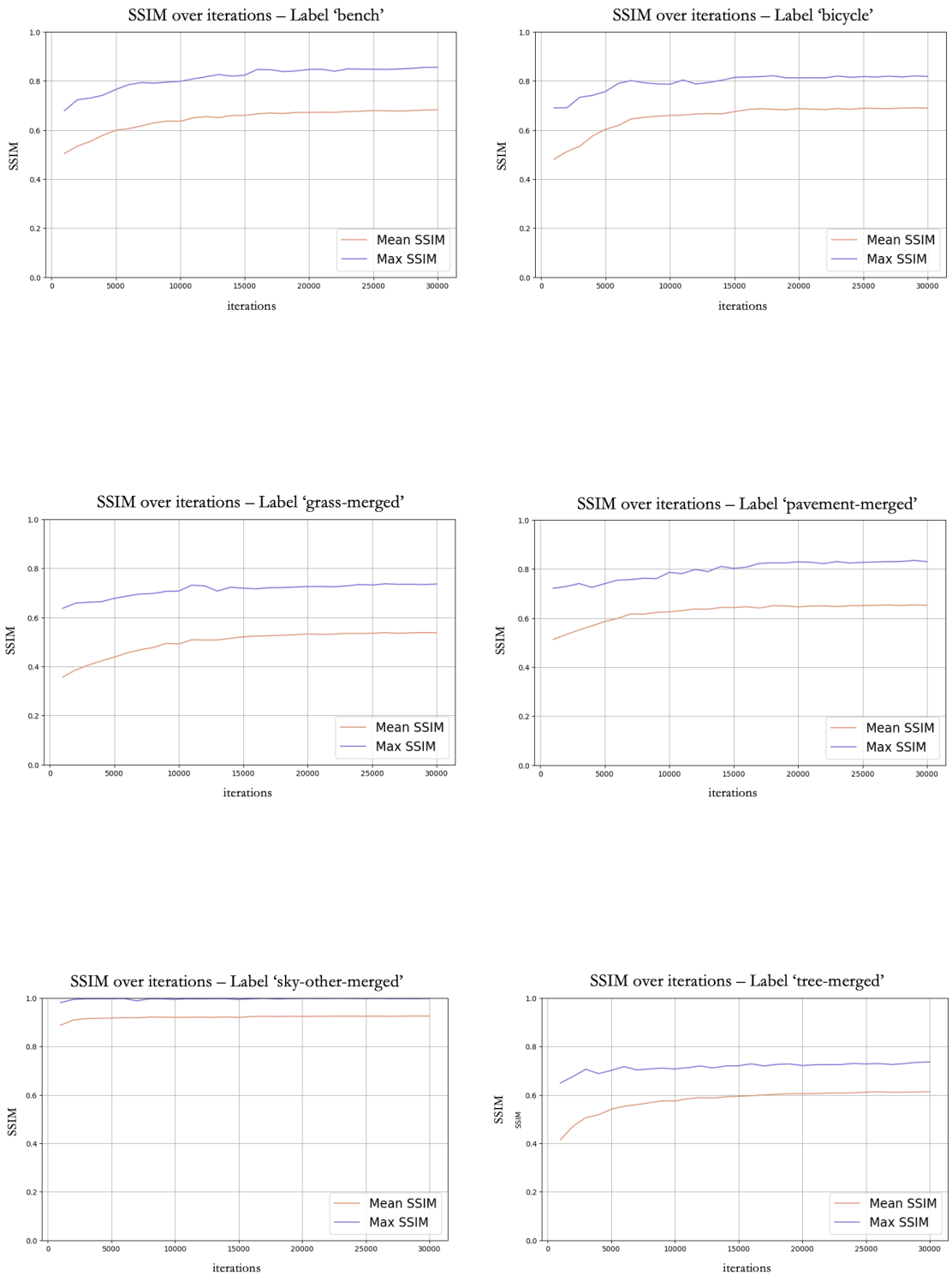
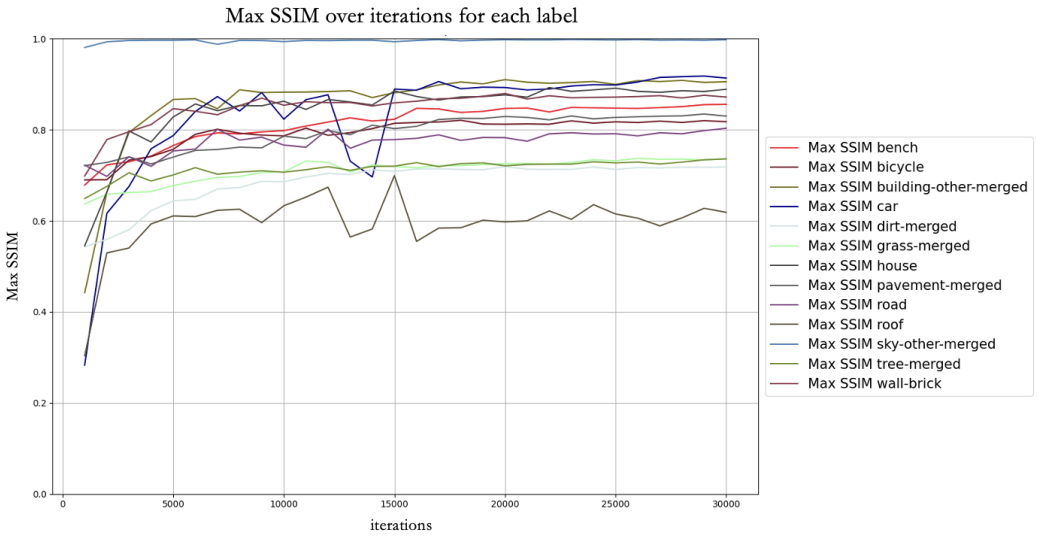
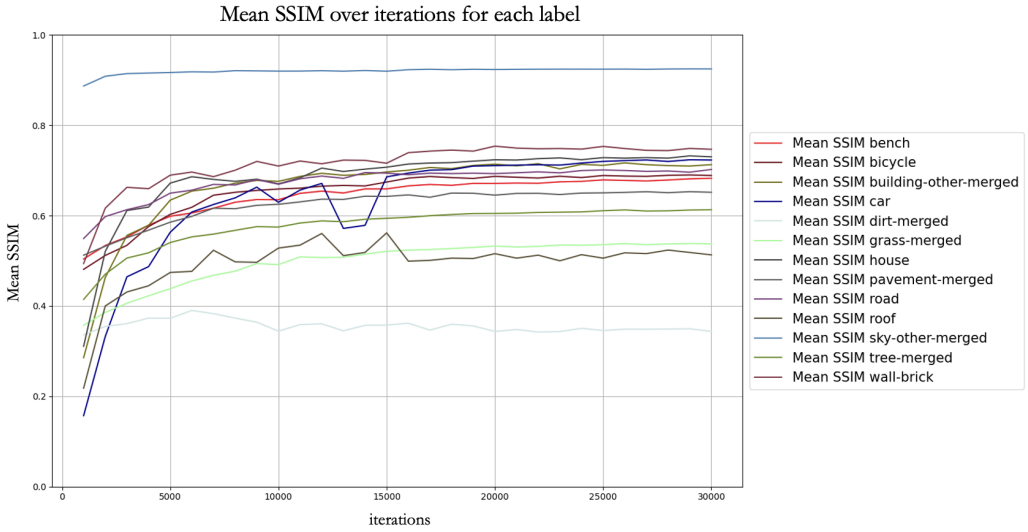


Figure 6.6: Mean and Max SSIM over iteration for most frequest semantic classes



(a) Max SSIM



(b) Mean SSIM

Figure 6.7: SSIM over iterations for the semantic labels of `bicycle` scene. (a) represents the maximum SSIM achieved for each label, while (b) shows the mean SSIM across the same labels. Both graphs highlight the performance improvement during training, with most labels reaching convergence between 10,000 and 20,000 iterations. Differences in SSIM values across labels reflect the varying complexity of the objects in the scene.

6.3 SSIM FITTING RESULTS

In this section, we present a mathematical model that expresses SSIM as a function of both the number of iterations and the minimum distance from a reference point. To determine the coefficients of this relationship, we performed a fitting analysis for each semantic class based on two distinct curves: SSIM as a function of iterations and SSIM as a function of minimum distance.

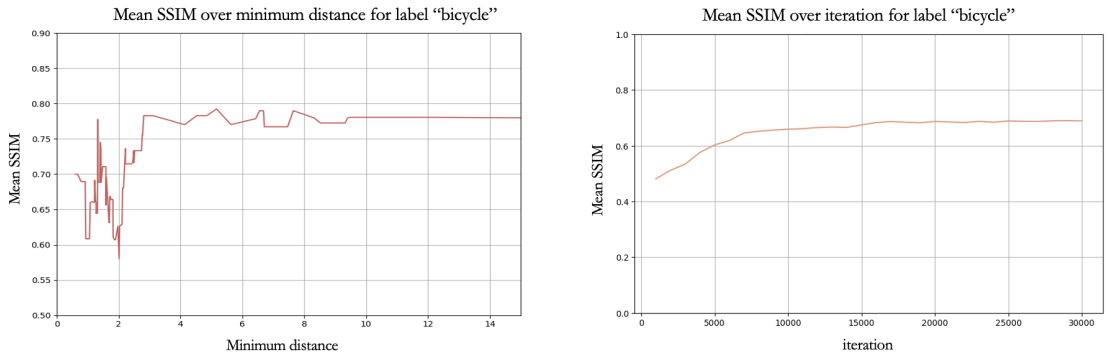


Figure 6.8: Mean SSIM as a function of minimum distance and number of iterations for the "bicycle" label. The left plot shows the variation of SSIM with respect to the minimum distance, while the right plot depicts the evolution of SSIM over the number of iterations. These curves will be used for fitting models to capture the relationship between SSIM, spatial characteristics, and optimization iterations

The relationship between SSIM and the number of iterations is described by a logarithmic function:

$$\text{SSIM}(i, d_{min} = \text{cost}) = a \cdot \log(i) + b \tag{6.1}$$

where i is the iteration and a is a fitting coefficient that determines the steepness of the curve, indicating how rapidly SSIM improves as the number of iterations increases. On the other hand, the relationship between SSIM and the minimum distance d_{min} is modeled using a piecewise function defined as follows:

$$\text{SSIM}(i = \text{cost}, d_{min}) = \begin{cases} C_1 \cdot e^{-\gamma_1 |d_{min} - \mu_1|^{\alpha_1}} & \text{if } d_{min} < \beta \\ C_2 \cdot e^{-\gamma_2 |d_{min} - \mu_2|^{\alpha_2}} & \text{if } d_{min} \geq \beta \end{cases} \tag{6.2}$$

In this piecewise model, C_1 and C_2 are scaling coefficients for the SSIM values, while γ_1 and γ_2 control the rate at which SSIM decays as a function of d_{min}. The parameters μ_1 and μ_2 represent the reference points for d_{min}, and α_1 and α_2 define the shape of the decay. The threshold β determines when the function switches between the two parts.

Since our goal is to derive a unified equation for SSIM that incorporates both iteration and distance, we assume that the distance-related information from Equation (6.1) is

encapsulated in the coefficient a. Thus, we integrate the relationship between SSIM and d_{min}, as described by Equation (6.2), into a.

$$a = K_n \cdot C_n e^{-\gamma_n |d_{min} - \mu_n|^{\alpha_n}} \qquad \text{with} \qquad n = 1, 2 \tag{6.3}$$

where K_n relates the two estimations. After fitting the SSIM vs. d_{min} curve to estimate the coefficients and the threshold β, we obtain the final fitting function:

$$\text{SSIM}(i, d_{min}) = \begin{cases} K_1 \cdot e^{-\gamma_1 |d_{min} - \mu_1|^{\alpha_1}} \cdot \log(i) + b & \text{if } d_{min} < \beta \\ K_2 \cdot e^{-\gamma_2 |d_{min} - \mu_2|^{\alpha_2}} \cdot \log(i) + b & \text{if } d_{min} \geq \beta \end{cases} \tag{6.4}$$

By fitting this model to the SSIM data for each semantic classes, we capture the underlying trends that describe how visual quality evolves with respect to both iteration count and the spatial characteristics of the scene. By setting a target SSIM value and knowing the camera position from which the scene is viewed, and therefore the distances of objects from the camera, this model allows us to select the optimal iteration that provides the appropriate visual quality for each object. The final scene is then composed of all objects, each rendered at their respective optimal iterations.

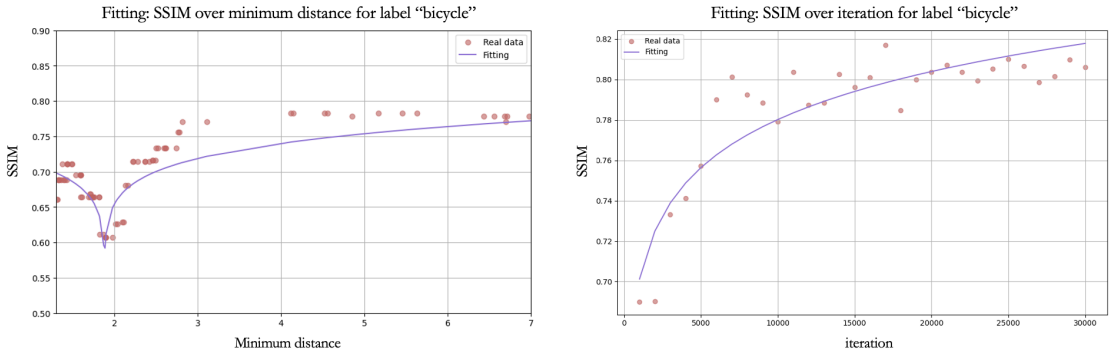


Figure 6.9: Fitting SSIM for label "bicycle" over minimum distance (left) and iteration count (right). In both cases, the red points represent real data, and the purple line corresponds to the fitted curve with the described equation. In this case, the coefficients obtained from the fitting process are $C_1 : 1.0$, $\gamma_1 : 1.4269$, $\mu_1 : 3.1625$, $\alpha_1 : 2.4362$, $C_2 : 0.3426$, $\gamma_2 : -0.7355$, $\mu_2 : 1.8787$, $\alpha_2 : 0.0606$, $\beta : 2.0$, $a : 0.0342$, $b : 0.4645$.

7

Conclusion

This thesis has explored the application of 3D Gaussian Splatting in the context of real-time visualization for eXtended Reality environments. The objective of the research was to address the challenges of achieving high visual fidelity and computational efficiency on devices with limited resources, such as mobile and wearable XR systems. To achieve this, a semantic-based Level of Detail system was developed and integrated into the rendering process. This system dynamically adapts the complexity of the 3D scene based on the user's position and the semantic importance of objects within the environment, allowing for a more efficient allocation of computational resources while preserving a high-quality user experience.

The proposed method leverages 3DGS to represent the scene's geometry and appearance through a set of spatially distributed Gaussians. The flexibility and differentiability of this representation offer significant advantages over traditional rendering techniques, enabling real-time performance without compromising the richness of the visual output. By incorporating semantic segmentation, the system prioritizes objects for high-resolution rendering based on their relevance in the scene, while less important elements are rendered with lower detail. This approach reduces the computational burden, particularly in regions of the scene that are outside the user's focus, thereby optimizing the rendering process.

Semantic segmentation played an important role in guiding the optimization process, ensuring that key areas were rendered in greater detail without affecting the overall performance of the system. The findings suggest that this approach can enhance the responsiveness and immersion of XR applications, making it a promising solution for real-time visualization on resource-constrained platforms.

7.1 Future Works

The research presented in this thesis highlights the potential of 3D Gaussian Splatting as a powerful tool for improving the efficiency and quality of real-time rendering in XR environments. The adaptive LOD system developed in this work provides a novel way to manage scene complexity based on both user position and semantic importance, with the goal of ensuring a seamless and immersive experience. While the project is still in progress, a comprehensive evaluation will be conducted following the described methodology. Future research could focus on refining the optimization process by incorporating additional metrics or leveraging deep learning techniques to predict optimal iterations. Furthermore, methods such as motion-based prioritization or user attention tracking could be explored to further improve the system's adaptability. Expanding the application of this adaptive 3DGS to more complex and dynamic environments, as well as testing it on larger or synthetic datasets, could be valuable for evaluating the model's generalization capabilities. Finally, these strategies could be integrated into an XR application to conduct practical tests and assess real-world performance.

Bibliography

[1] Kara-Ali Aliev et al. *Neural Point-Based Graphics*. 2020. arXiv: 1906.08240 [cs.CV]. URL: https://arxiv.org/abs/1906.08240.

[2] Jonathan T. Barron et al. *Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields*. 2022. arXiv: 2111.12077 [cs.CV]. URL: https://arxiv.org/abs/2111.12077.

[3] Filip Biljecki, Hugo Ledoux, and Jantien Stoter. "Redefining the Level of Detail for 3D models". In: *GIM International* 28 (Nov. 2014), pp. 21–23.

[4] B. Brand, M. Bätz, and Joachim Keinert. "CAMORPH: A TOOLBOX FOR CONVERSION BETWEEN CAMERA PARAMETER CONVENTIONS". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLVIII-2/W1-2022 (Dec. 2022), pp. 29–36. DOI: 10.5194/isprs-archives-XLVIII-2-W1-2022-29-2022.

[5] Elena Camuffo, Daniele Mari, and Simone Milani. "Recent Advancements in Learning Algorithms for Point Clouds: An Updated Overview". In: *Sensors* 22.4 (2022). ISSN: 1424-8220.

[6] Elena Camuffo et al. "Deep 3D Model Optimization for Immersive and Interactive Applications". In: *2022 10th European Workshop on Visual Information Processing (EUVIP)*. IEEE. 2022, pp. 1–6.

[7] Guikun Chen and Wenguan Wang. *A Survey on 3D Gaussian Splatting*. 2024. arXiv: 2401.03890 [cs.CV]. URL: https://arxiv.org/abs/2401.03890.

[8] Liang-Chieh Chen et al. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2017. arXiv: 1606.00915 [cs.CV]. URL: https://arxiv.org/abs/1606.00915.

[9] Liang-Chieh Chen et al. *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*. 2016. arXiv: 1412.7062 [cs.CV]. URL: https://arxiv.org/abs/1412.7062.

[10] Jayfus T. Doswell and Anna Skinner. "Augmenting Human Cognition with Adaptive Augmented Reality". In: *Foundations of Augmented Cognition. Advancing Human Performance and Decision-Making through Adaptive Systems*. 2014, pp. 104–113.

[11] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. "Volume rendering". In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '88. New York, NY, USA: Association for Computing Machinery, 1988, pp. 65–74. ISBN: 0897912756. DOI: 10.1145/54852.378484. URL: https://doi.org/10.1145/54852.378484.

[12] Tho Nguyen Duc et al. "Convolutional Neural Networks for Continuous QoE Prediction in Video Streaming Services". In: *IEEE Access* 8 (2020), pp. 116268–116278.

[13] Matteo Gadaleta et al. "D-DASH: A Deep Q-Learning Framework for DASH Video Streaming". In: *IEEE Transactions on Cognitive Communications and Networking* 3.4 (2017), pp. 703–718.

[14] Kyle Gao et al. *NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review*. 2023. arXiv: 2210.00379 [cs.CV]. URL: https://arxiv.org/abs/2210.00379.

[15] Quankai Gao et al. *GaussianFlow: Splatting Gaussian Dynamics for 4D Content Creation*. 2024. arXiv: 2403.12365 [cs.CV]. URL: https://arxiv.org/abs/2403.12365.

[16] Marcel Geppert et al. "Privacy Preserving Structure-from-Motion". In: *ECCV*. 2020.

[17] Antoine Guédon and Vincent Lepetit. *SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering*. 2023. arXiv: 2311.12775 [cs.GR]. URL: https://arxiv.org/abs/2311.12775.

[18] Jon Hasselgren et al. "Appearance-Driven Automatic 3D Model Simplification". In: *Proc. of Eurographics Symposium on Rendering 2021*. Apr. 2021.

[19] Peter Hedman et al. "Deep Blending for Free-viewpoint Image-based Rendering". In: 37.6 (2018), 257:1–257:15.

[20] Binbin Huang et al. "2D Gaussian Splatting for Geometrically Accurate Radiance Fields". In: *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24*. SIGGRAPH '24. ACM, July 2024. DOI: 10.1145/3641519.3657428. URL: http://dx.doi.org/10.1145/3641519.3657428.

[21] Te-Yuan Huang et al. "A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service". In: *SIGCOMM Comput. Commun. Rev.* 44.4 (Aug. 2014), pp. 187–198.

[22] James T. Kajiya. "The rendering equation". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: https://doi.org/10.1145/15886.15902.

[23] Hiroharu Kato et al. *Differentiable Rendering: A Survey*. 2020. arXiv: 2006.12057 [cs.CV]. URL: https://arxiv.org/abs/2006.12057.

[24] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4 (July 2023).

[25] Hakran Kim, Yongik Yoon, and Hwajin Park. "Adaptation Method for Level of Detail (LOD) of 3Dcontents". In: *2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007)*. 2007, pp. 879–884.

[26] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[28] Samuli Laine and Tero Karras. "High-performance software rasterization on GPUs". In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2011, pp. 79–88. ISBN: 9781450308960. DOI: 10.1145/2018323.2018337. URL: https://doi.org/10.1145/2018323.2018337.

[29] Xiaohan Lei et al. *GaussNav: Gaussian Splatting for Visual Navigation*. 2024. arXiv: 2403.11625 [cs.CV]. URL: https://arxiv.org/abs/2403.11625.

[30] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312.

[31] Jie Liu, Xiaoming Tao, and Jianhua Lu. "QoE-Oriented Rate Adaptation for DASH With Enhanced Deep Q-Learning". In: *IEEE Access* 7 (2019), pp. 8454–8469.

[32] Tao Lu et al. *Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering*. 2023. arXiv: 2312.00109 [cs.CV]. URL: https://arxiv.org/abs/2312.00109.

[33] Ben Mildenhall et al. "NeRF: representing scenes as neural radiance fields for view synthesis". In: *Commun. ACM* 65.1 (Dec. 2021), pp. 99–106. ISSN: 0001-0782.

[34] Thomas Müller et al. "Instant neural graphics primitives with a multiresolution hash encoding". In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

[35] Dang H Nguyen et al. "An adaptive method for low-delay 360 VR video streaming over HTTP/2". In: *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*. 2018, pp. 261–266.

[36] Ozgur Oyman and Sarabjot Singh. "Quality of experience for HTTP adaptive streaming services". In: *IEEE Communications Magazine* 50.4 (2012), pp. 20–27.

[37] AKM Shahariar Azad Rabby and Chengcui Zhang. *BeyondPixels: A Comprehensive Review of the Evolution of Neural Radiance Fields*. 2024. arXiv: 2306.03000 [cs.CV]. URL: https://arxiv.org/abs/2306.03000.

[38] Darius Rückert, Linus Franke, and Marc Stamminger. "ADOP: approximate differentiable one-pixel point rendering". In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530122. URL: https://doi.org/10.1145/3528223.3530122.

[39] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[40] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. *Software Rasterization of 2 Billion Points in Real Time*. 2022. arXiv: 2204.01287 [cs.GR]. URL: https://arxiv.org/abs/2204.01287.

[41] Jinseok Seo, Gerard Jounghyun Kim, and Kyo Chul Kang. "Levels of detail (LOD) engineering of VR objects". In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. VRST '99. London, United Kingdom: Association for Computing Machinery, 1999, pp. 104–110. ISBN: 1581131410. DOI: 10.1145/323663.323680. URL: https://doi.org/10.1145/323663.323680.

[42] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 2014).

[43] Iraj Sodagar. "The MPEG-DASH Standard for Multimedia Streaming Over the Internet". In: *IEEE MultiMedia* 18.4 (2011), pp. 62–67.

[44] Jiaxiang Tang et al. *DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation*. 2024. arXiv: 2309.16653 [cs.CV]. URL: https://arxiv.org/abs/2309.16653.

[45] Babak Taraghi et al. "Intense: In-Depth Studies on Stall Events and Quality Switches and Their Impact on the Quality of Experience in HTTP Adaptive Streaming". In: *IEEE Access* 9 (2021), pp. 118087–118098.

[46] A. Tewari et al. "State of the Art on Neural Rendering". In: *Computer Graphics Forum* 39.2 (2020), pp. 701–727. DOI: https://doi.org/10.1111/cgf.14022. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14022. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14022.

[47] *Toy Gaussian Splatting visualization in Unity*. 2023. URL: https://github.com/aras-p/UnityGaussianSplatting.

[48] *Unity Real-Time Development Platform*. URL: https://unity.com.

[49] Neil Vaughan, Bodgan Gabrys, and Venketesh N. Dubey. "An overview of self-adaptive technologies within virtual reality training". In: *Computer Science Review* 22 (2016), pp. 65–87.

[50] Cong Wang et al. "Design and Analysis of QoE-Aware Quality Adaptation for DASH: A Spectrum-Based Approach". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 13.3s (July 2017).

[51] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[52] Mark Weber et al. *DeepLab2: A TensorFlow Library for Deep Labeling*. 2021. arXiv: 2106.09748 [cs.CV]. URL: https://arxiv.org/abs/2106.09748.

[53] Tianyi Xie et al. *PhysGaussian: Physics-Integrated 3D Gaussians for Generative Dynamics*. 2024. arXiv: 2311.12198 [cs.GR]. URL: https://arxiv.org/abs/2311.12198.

[54] Chi Yan et al. *GS-SLAM: Dense Visual SLAM with 3D Gaussian Splatting*. 2024. arXiv: 2311.11700 [cs.CV]. URL: https://arxiv.org/abs/2311.11700.

[55] Alex Yu et al. *Plenoxels: Radiance Fields without Neural Networks*. 2021. arXiv: 2112.05131 [cs.CV]. URL: https://arxiv.org/abs/2112.05131.

[56] Richard Zhang et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 586–595. DOI: 10.1109/CVPR.2018.00068.

Acknowledgments

Vorrei ringraziare tutte le persone che mi hanno accompagnato e sostenuto durante questo viaggio, rendendo possibile il raggiungimento di questo traguardo.

In particolare vorrei ringraziare Alessio che mi è stato sempre accanto in modo incondizionato (seguendomi persino a Barcellona), dandomi la forza di non mollare nemmeno nei momenti di maggiore sconforto e assecondando le mie folli idee. Grazie per aver condiviso con me le gioie e le fatiche di questo percorso.

Grazie di cuore ai miei genitori che sono sempre stati per me un punto di riferimento. Grazie per non avermi mai fatto mancare il vostro supporto e per avermi dato la libertà di prendere le mie decisioni, incoraggiandomi a inseguire i miei sogni. Un grazie speciale va a Gaia e Giulia, che con le loro risate e la loro spensieratezza hanno reso le mie giornate più leggere e colorate. Un ringraziamento anche ai miei nonni che si sono sempre interessati ai miei studi e alle mie passioni.

Vorrei ringraziare tutti i miei amici, che negli anni hanno ascoltato le mie lamentele e con cui ho collezionato bellissimi ricordi. Grazie a Lavinia e Manuele, compagni di studio e progetti, per gli innumerevoli zoom e l'aiuto reciproco: senza di voi, questi anni non sarebbero stati gli stessi. Grazie anche a Luca e a Pietro per essere sempre stati disponibili nel momento del bisogno.

Un grazie sentito al mio relatore, il professor Milani per aver creduto in me, per le opportunità che mi ha offerto e per la pazienza. Ringrazio anche il professor Badia, che ha scelto di darmi fiducia, offrendomi la possibilità di lavorare nel mondo della ricerca. Grazie infinite ad Elena che è stata un sostegno costante e paziente. Grazie per i tuoi consigli preziosi, per la tua disponibilità e per tutto l'aiuto che mi hai offerto.