Friedrich-Alexander-Universität
Faculty of Engineering

Doctoral thesis

2025

Linus Franke

Neural Point-based Rendering for Immersive Novel View Synthesis

Neural Point-based Rendering for Immersive Novel View Synthesis

---

Neuronales Punktbasiertes Rendering für Immersive Neuansichtssynthese

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur Erlangung des Doktorgrades

Doktor–Ingenieur (Dr.–Ing.)

vorgelegt von

Linus Rasmus Leonard Franke

Als Dissertation genehmigt von
der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen–Nürnberg

Tag der mündlichen Prüfung:	26.05.2025
Gutachter/in:	Prof. Dr.-Ing. Marc Stamminger
	Prof. Dr.-Ing. Justus Thies

To my father, forever in my heart.

Abstract

Recent advances in neural rendering have greatly improved the realism and efficiency of digitizing real-world environments, enabling new possibilities for virtual experiences. However, achieving high-quality digital replicas of physical spaces is challenging due to the need for advanced 3D reconstruction and real-time rendering techniques, with visual outputs often deteriorating in challenging capturing conditions. Thus, this thesis explores point-based neural rendering approaches to address key challenges such as geometric inconsistencies, scalability, and perceptual fidelity, ultimately enabling realistic and interactive virtual scene exploration. The vision here is to enable immersive virtual reality (VR) scene exploration and virtual teleportation with the best perceptual quality for the user.

This work introduces techniques to improve point-based Novel View Synthesis (NVS) by refining geometric accuracy and reducing visual artifacts. By detecting and correcting errors in point-cloud-based reconstructions, this approach improves rendering stability and accuracy. Additionally, an efficient rendering pipeline is proposed that combines rasterization with neural refinement to achieve high-quality results at real-time frame rates, ensuring smooth and consistent visual output across diverse scenes.

To extend the scalability of neural point representations, a hierarchical structure is presented that efficiently organizes and renders massive point clouds, enabling real-time NVS of city-scale environments. Furthermore, a perceptually optimized foveated rendering technique is developed for VR applications, leveraging the characteristics of the human visual system to balance performance and perceptual quality. Lastly, a real-time neural reconstruction technique is proposed that eliminates preprocessing requirements, allowing for immediate virtual teleportation and interactive scene exploration.

Through these advances, this thesis pushes the boundaries of neural point-based rendering, offering solutions that balance quality, efficiency, and scalability. The findings pave the way for more interactive and immersive virtual experiences, with applications spanning VR, augmented reality (AR), and digital content exploration.

Zusammenfassung

Jüngste Fortschritte im Bereich des neuronalen Renderings haben den Realismus und die Effizienz der Digitalisierung realer Umgebungen erheblich verbessert und eröffnen neue Möglichkeiten für immersive virtuelle Erlebnisse. Die Erstellung hochwertiger digitaler Repliken physischer Räume stellt eine Herausforderung dar. Grund dafür ist, dass fortschrittliche Techniken der 3D-Rekonstruktion und des Echtzeit-Renderings erforderlich sind, wobei visuelle Ausgaben häufig von schwierigen Aufnahmebedingungen beeinträchtigt werden. Diese Dissertation untersucht daher punktbasierte neuronale Rendering-Ansätze, um zentrale Herausforderungen wie geometrische Inkonsistenzen, Skalierbarkeit und wahrgenommene visuelle Treue zu adressieren, und ermöglicht so eine realistische und interaktive virtuelle Szenenerkundung. Die Vision besteht darin, immersive Szenenerkundung für virtuelle Realitäten (VR) und virtuelle Teleportation mit der bestmöglichen wahrgenommenen Qualität für den Benutzer zu ermöglichen.

Diese Arbeit stellt Techniken vor, um die punktbasierte Synthese neuer Ansichten zu verbessern, indem die geometrische Genauigkeit verfeinert und visuelle Artefakte reduziert werden. Durch die Identifikation und Korrektur von Fehlern in punktwolkenbasierten Rekonstruktionen verbessert unser Ansatz die Rendering-Stabilität und Genauigkeit. Zudem schlagen wir eine effiziente Rendering-Pipeline vor, die Rasterisierung mit neuronaler Verfeinerung kombiniert, um hochwertige Ergebnisse mit Echtzeit-Bildraten zu erzielen und so eine gleichmäßige und konsistente visuelle Ausgabe über verschiedene Szenen hinweg zu gewährleisten.

Zur Erweiterung der Skalierbarkeit neuronaler Punktdarstellungen präsentieren wir eine hierarchische Struktur, die massive Punktwolken effizient organisiert und rendert, wodurch die Echtzeit-Visualisierung von großflächigen Umgebungen ermöglicht wird. Darüber hinaus wird eine wahrnehmungsoptimierte Rendering-Technik für VR-Anwendungen entwickelt, die die Eigenschaften des menschlichen visuellen Systems nutzt, um Leistung und wahrgenommene Qualität auszubalancieren. Abschließend schlagen wir eine Echtzeit-neuronale Rekonstruktionstechnik vor, die Vorverarbeitungsanforderungen eliminiert und so sofortige virtuelle Teleportation und interaktive Szenenerkundung ermöglicht.

Durch diese Fortschritte erweitert diese Dissertation die Grenzen des neuronalen punktbasierten Renderings und bietet Lösungen, die Qualität, Effizienz und Skalierbarkeit in Einklang bringen. Unsere Ergebnisse ebnen den Weg für interaktive und immersive virtuelle Erlebnisse, mit Anwendungen in den Bereichen VR, augmentierte Realität (AR) und digitale Inhaltserschließung.

Acknowledgments

While pursuing a doctorate is largely a solitary task, I owe the existence of this thesis to the support of many, whom I would like to sincerely thank.

First and foremost, I am deeply grateful to my advisor, Marc Stamminger, for offering me the chance to pursue my doctoral studies, and for his invaluable insights, guidance, support, and encouragement throughout my research over the past decade, beginning from my undergraduate years. Marc, I have and always will admire your patience, fortitude, and dedication. I am truly grateful for the opportunity to learn under your mentorship! And thanks for all the advice with what comes afterwards.

Additionally, I would also like to thank my Bachelor thesis adviser Kai Selgrad for luring me into the world of academia. Thanks for your guidance early on and for inspiring me to go down the path of research. I am thoroughly enjoying it.

Furthermore, I am grateful to my lovely colleagues and lab mates, past and present, at the LGDV and the VCE for the countless discussions and collaborative efforts. I will miss all the coffee room discussions and something-breaks, and also the encouragement and focus during stressful deadlines. Your presence made my time at the chair truly unique. Special thanks to Laura for all the meaningful personal conversations but also for the shared projects, brainstorming sessions, and support during every deadline and challenge, and for pushing me whenever I needed to be pushed. And sorry for the bad timing with deadlines during the retreat! I also want to thank Darius for all the advice about everything points; I fully enjoyed every discussion. And thanks to Mathias for the always interesting project discussions and bantering about everything else. Furthermore, I am grateful for Niko, Noah, and Laura for being awesome office mates. Also, thanks to Bernhard, Tim, and Tobias for always offering additional advice, having an open ear for my questions and helping me with the next steps.

Next, all thanks to my amazing co-authors of the papers presented in this thesis. Marc, Laura, Darius, Matthias, Mathias, Tim, and Noah, thank you for your support, guidance, and hard work. Also, thanks to Kai, Alex, Niko, Jana, Svenja, Joachim, Florian, Fabian, Moritz, Susanna, Martin, Marcus, Lukas, Josef, Max, and Bernhard for collaborating on different projects and allowing me to learn from you.

Moreover, I would like to express my gratitude for the assistance from NavVis GmbH, specifically Matthias, Stefan, Michael, and Tim, who always patiently listened to me during project meetings and always provided insightful comments and discussions. Thank you so much for your support on this journey.

To my family and friends, thank you for your endless encouragement and understanding. Your belief in me was a constant source of motivation. Special thanks to my mother, Edith, for your unwavering support and for always listening to me ramble about research.

Last but not least, I want to thank my partner Sarah for always being there for me, during every long workday and while writing this thesis. Also, thank you for your encouragement and patience with me during stressful deadlines and when I had to follow a crazy research idea. Your support means the world to me. Without you, this thesis and the doctorate as a whole would not have been possible.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, AI technologies were used to assist in the writing process. Specifically, *ChatGPT* (GPT-4o, mini) (OpenAI L.L.C., San Francisco, CA, USA and OpenAI Ireland Ltd., Dublin, Ireland) and *Writefull* (ThinqLab Ltd, London, UK) were used in order to assist with rephrasing and improving readability. After using these tools, the manuscript was carefully reviewed and the content was edited as needed. No tools or services were used for content generation. The content of this dissertation was entirely created by the author, who has ensured that all material presented reflects their original work and research. The author takes full responsibility for the content of the dissertation.

Contents

List of Figures

List of Tables

Acronyms

3DGS	3D Gaussian Splatting. 20, 21, 22, 39, 40, 45, 50, 51, 61, 67, 68, 71, 72, 75, 76, 77, 80, 81
AR	Augmented Reality. 1, 4, 98
CNN	Convolutional Neural Network. 9, 13, 17, 18, 19, 20, 54, 58, 68, 69, 71, 73, 74, 75, 80
GAN	Generative Adversarial Network. 20, 98
IBR	Image-based Rendering. 7, 8, 13, 14, 15, 16, 20, 22, 84, 85
IMU	Inertial Measurement Unit. 11, 12, 60
LiDAR	Light Detection and Ranging. 1, 2, 10, 11, 23, 28, 41, 45, 53, 54, 58, 59, 60, 62, 64, 83, 84, 85, 86, 88, 89, 90, 93, 94
LoD	Level of Detail. 5, 21, 54, 55, 56, 57, 60, 62, 64, 66
MLP	Multilayer Perceptron. 15, 16, 17, 20, 70
MVS	Multi-view Stereo. 8, 9, 17, 18, 20, 23, 24, 29, 41, 45, 54, 65, 71, 88, 89
NeRF	Neural Radiance Field. 15, 16, 17, 21, 22, 24, 55, 70, 71, 76, 92
NVS	Novel View Synthesis. 1, 4, 5, 7, 10, 11, 13, 14, 16, 17, 19, 20, 21, 22, 34, 35, 53, 54, 56, 59, 84, 92, 93, 95, 97, 98
SfM	Structure-from-Motion. 8, 9, 17, 20, 21, 50, 71
SLAM	Simultaneous Localization and Mapping. 11, 60, 83, 85, 93
VR	Virtual Reality. 1, 2, 4, 5, 19, 22, 51, 66, 67, 68, 69, 70, 71, 72, 76, 77, 78, 80, 81, 97, 98

CHAPTER 1
Introduction

Interactive exploration in digital replication of real-world environments have long been fundamental goals in computer graphics and computer vision. Advances in these fields have led to increasingly realistic virtual representations [Tewari et al. 2022], enabling applications in gaming, Virtual Reality (VR), urban planning, and autonomous navigation. However, achieving high-fidelity digital twins of physical spaces remains a complex challenge, requiring sophisticated techniques for reconstructing digital 3D representations from captured real-world data and for the real-time rendering of them. Despite significant progress, key issues persist in capturing fine-grained details, ensuring computational efficiency on large-scale captures, and maintaining consistent rendering for the complex human visual system.

To achieve this goal, digital scene reconstruction and rendering form the two core stages of the pipeline. In the reconstruction phase, a digital model of the environment is generated by capturing images using video or photo cameras, often supplemented with additional sensing modalities such as Light Detection and Ranging (LiDAR) [McManamon 2019] or depth cameras [Zollhöfer et al. 2018] to provide distance information. These images are then aligned with one another through registration techniques [Schönberger and Frahm 2016] and subsequently processed into a coherent scene representation that integrates both geometric structure and appearance, ensuring an accurate digital replica of the physical world [Schönberger et al. 2016].

After constructing the scene representation, it is possible to render it from new angles that were not part of the initial capture, a process called Novel View Synthesis (NVS) [Shum et al. 2008]. This allows for a flexible and engaging viewing experience, letting users examine the digital environment from various positions. To preserve realism, the rendering must produce visually consistent and high-fidelity outputs. Furthermore, for interactive applications where users can navigate the scene by altering their position and viewpoint in real-time, high rendering performance is essential to provide smooth, low-latency feedback. Looking at VR, where a user is fully immersed in the digital world through head-mounted displays, ensuring a seamless and responsive experience is vital. Similarly, in Augmented Reality (AR), where digital elements are overlayed with the real world, the same is true with the additional constraint of ensuring realism in displayed elements.

A promising approach to improve the reconstruction-to-rendering pipeline is differentiable rendering [Kato et al. 2020]. Unlike traditional rendering pipelines, which operate in a feedforward manner, differentiable rendering allows gradients to propagate through the rendering process, making it possible to refine scene representations based on renderings of observed images. This capability is particularly valuable for inverse problems, such as optimizing geometric structures, materials, and capture positions to achieve a more accurate reconstruction. By integrating differentiable rendering, it becomes possible to iteratively refine digital models, improve consistency between captured and rendered views, and enable self-supervised learning approaches for scene understanding. Consequently, differentiable rendering plays a crucial role in the advancement of realistic, high-fidelity digital representations that support interactive exploration and analysis.

This thesis aims to enable VR-ready scene exploration and virtual teleportation from the goal-oriented perspective of arriving at the best perceptual quality for the user. However, several challenges must be addressed to achieve this objective. Inaccuracies in scanning, image registration, and material representation can lead to visual artifacts and misalignments, undermining realism. Moreover, to align

with the characteristics of the human visual system, the rendering process must ensure smooth motion, proper depth perception, and perceptual continuity, preventing distractions such as flickering or abrupt changes in appearance. Furthermore, rendering performance must be optimized to deliver low-latency real-time feedback, which is particularly critical for immersive VR experiences. This can be especially challenging in large-scale urban scenarios, where users explore whole cities. Finally, generalization across diverse scenes remains a significant challenge, as the pipeline must adapt to varying geometries and textures without extensive per-scene fine-tuning.

In this work, we propose a point-based representation for both the reconstruction and rendering of digital environments. This representation is grounded in a Lagrangian formulation, providing an explicit and manipulable structure for scene modeling. The points used in this approach can be derived from sources such as image-to-image point matching or LiDAR scans, offering a flexible means of capturing scene geometry. However, while this point-based representation effectively captures the broad structure of the environment, it fundamentally misses finer details — issues that we address through advanced refinement techniques.

Building on this foundation, we extend the basic point primitive with a trilinear splatting formulation that facilitates accurate differentiable refinement of both geometry and camera poses, allowing for more precise scene optimization. This method enables iterative adjustments to the scene representation, improving both its accuracy and consistency with real-world data. Furthermore, our point-based approach is scalable and capable of handling large-scale environments that involve hundreds of millions of points by extending it with hierarchical rendering and optimization structures. This scalability ensures that the representation remains effective even for complex city-scale scenes.

The flexibility of the point-based primitives also allows for seamless integration with the properties of the human visual system, improving the perceptual coherence of the rendered environments. This integration ensures that the real-time rendering process aligns more closely with how humans perceive visual stimuli, enhancing realism and reducing perceptual discrepancies. Additionally, extensions support ad hoc rendering during the capture process, which eliminates the need for extensive preprocessing. This capability allows for more immediate rendering results, enabling users to explore the virtual environment immediately in real-time as capturing is performed.

Publications by the Author for this Thesis

This thesis is based on the following publications ([†] denotes equal contribution):

[Franke et al. 2023] **Linus Franke**, Darius Rückert, Laura Fink, Matthias Innmann, and Marc Stamminger. 2023. *VET: Visual Error Tomography for Point Cloud Completion and High-Quality Neural Rendering*. In SIGGRAPH Asia 2023 Conference Papers (Sydney, NSW, Australia) (SA '23). Association for Computing Machinery, New York, NY, USA, Article 38, 12 pages. https://doi.org/10.1145/3610548.3618212

[Harrer et al. 2023] Mathias Harrer[†], **Linus Franke**[†], Laura Fink, Marc Stamminger, and Tim Weyrich. 2023. *Inovis: Instant Novel-View Synthesis.* In SIGGRAPH Asia 2023 Conference Papers (Sydney, NSW, Australia) (SA '23). Association for Computing Machinery, New York, NY, USA, Article 14, 12 pages. https://doi.org/10.1145/3610548.3618216

[Franke et al. 2024] **Linus Franke**, Darius Rückert, Laura Fink, and Marc Stamminger. 2024. *TRIPS: Trilinear Point Splatting for Real-Time Radiance Field Rendering.* Computer Graphics Forum 43, 2 (2024), p. e15012. https://doi.org/10.1111/cgf.15012

[Franke et al. 2025] **Linus Franke**, Laura Fink, and Marc Stamminger. 2025. *VR-Splatting: Foveated Radiance Field Rendering via 3D Gaussian Splatting and Neural Points.* To appear in Proceedings of the ACM on Computer Graphics and Interactive Techniques (i3D 2025) 8, 18, (May 2025), https://arxiv.org/abs/2410.17932

[Lewis et al. 2025] Noah Lewis, Darius Rückert, Marc Stamminger and **Linus Franke**. 2025. *NePO: Neural Point Octrees for Large-scale Novel View Synthesis.* 2025, currently submitted.

For Franke et al. [2023], Franke et al. [2024] and Franke et al. [2025], the author of this thesis contributed the algorithmic ideas and their implementations. Furthermore, he evaluated the proposed methods and wrote major parts of the paper.

Harrer and Franke et al. [2023] is the result of a supervised Master's thesis. The author of this thesis contributed the project idea, algorithmic idea, and supervised Mathias Harrer during the implementation in their thesis. In addition, the author of this thesis contributed in part to generating results for related methods and wrote the main parts of the paper.

Lewis et al. [2025] is the result of a supervised Master's thesis. The author contributed the project conceptualization, algorithmic idea, and supervised Noah Lewis during the implementation in their thesis. Furthermore, the author of this thesis evaluated parts of the proposed method and competing methods and wrote the main parts of the paper.

Additional Publications with Contributions by the Author

The following publications were first and coauthored before and during the doctoral studies, but are not subject of this thesis.

[Selgrad et al. 2016] Kai Selgrad, **Linus Franke**, and Marc Stamminger. 2016. *Tiled Depth of Field Splatting.* In Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Posters (Lisbon, Portugal) (EG '16). Eurographics Association, Goslar, DEU, 2 pages.

[Lier et al. 2016] Alexander Lier, **Linus Franke**, Marc Stamminger, and Kai Selgrad. 2016. *A Case Study in Implementation-Space Exploration.* In Proceedings of the 9th European Lisp Symposium on European Lisp Symposium (Kraków, Poland) (ELS2016). European Lisp Scientific Activities Association, Article 10, 8 pages.

[Franke et al. 2018] **Linus Franke**, Nikolai Hofmann, Marc Stamminger, and Kai Selgrad. 2018. *Multi-layer Depth of Field Rendering with Tiled Splatting.* Proceedings of the ACM on Computer Graphics and Interactive Techniques (i3D 2018) 1, 1, Article 6 (July 2018), 17 pages.

[Franke et al. 2021] **Linus Franke**, Laura Fink, Jana Martschinke, Kai Selgrad, and Marc Stamminger. 2021. *Time-Warped Foveated Rendering for Virtual Reality Headsets.* Computer Graphics Forum 40, 1 (2021), 14 pages.

[Rückert et al. 2022] Darius Rückert, **Linus Franke**, and Marc Stamminger. 2022. *ADOP: Approximate Differentiable One-Pixel Point Rendering.* ACM Trans. Graph. 41, 4, Article 99 (July 2022), 14 pages.

[Fink et al. 2023a] Laura Fink, Darius Rückert, **Linus Franke**, Joachim Keinert, and Marc Stamminger. 2023. *LiveNVS: Neural View Synthesis on Live RGB-D Streams.* In SIGGRAPH Asia 2023 Conference Papers (Sydney, NSW, Australia) (SA '23). Association for Computing Machinery, New York, NY, USA, Article 37, 11 pages.

[Fink et al. 2023b] Laura Fink, Svenja Strobel, **Linus Franke**, and Marc Stamminger. 2023. *Efficient Rendering for Light Field Displays using Tailored Projective Mappings.* Proceedings of the ACM on Computer Graphics and Interactive Techniques (i3D 2023) 6, 1, Article 3 (May 2023), 17 pages.

[Fink et al. 2024] Laura Fink, **Linus Franke**, Joachim Keinert, and Marc Stamminger. 2024. *Refinement of Monocular Depth Maps via Multi-View Differentiable Rendering.* arXiv preprint (arXiv:2410.03861)

[Hahlbohm et al. 2025a] Florian Hahlbohm, **Linus Franke**, Moritz Kappel, Susana Castillo, Martin Eisemann, Marc Stamminger, and Marcus Magnor. 2025. *INPC: Implicit Neural Point Clouds for Radiance Field Rendering.* 2025 International Conference on 3D Vision (3DV).

[Hahlbohm et al. 2025b] Florian Hahlbohm, Fabian Friederichs, Tim Weyrich, **Linus Franke**, Moritz Kappel, Susana Castillo, Marc Stamminger, Martin Eisemann, and Marcus Magnor. 2025. *Efficient Perspective-Correct 3D Gaussian Splatting Using Hybrid Transparency.* 2025. Computer Graphics Forum 44, 2.

Contribution of this Thesis

The vision of this thesis is to allow a user to realistically explore and teleport into real-world environments virtually, whether through VR, AR, or traditional desktop interfaces. Consequently, this work advances the current standards in neural differentiable rendering to facilitate these novel capabilities.

The vision entails numerous requirements, which we examine with a goal-driven approach. The human visual system is highly sensitive to rendering artifacts such as unnatural lighting, geometric errors, and temporal inconsistencies such as flickering [Weier et al. 2017]. These irregularities can greatly affect the perception of realism, as the human visual system is adept at detecting deviations from expected patterns. Achieving this with typical 3D reconstruction and rendering methods requires optimal conditions, as inaccuracies in the stages of capturing, reconstruction, or rendering lead to noticeable errors that propagate throughout the pipeline.

As such, in this thesis, we introduce methods to achieve high-fidelity VR-ready NVS, which refine the scene parameters efficiently for robust results in various challenging scenarios. We present a review of the background of these techniques in Chapter 2, which will also provide context for the terminologies used in the following description of contributions.

Our approach builds on neural point rendering pipelines [Aliev et al. 2020; Rückert et al. 2022], which typically rasterize a point cloud into pixel-sized splats, then process these sparse renderings with a neural network.

Chapter 3, based on the publication Franke et al. [2023], details our approach to resolving geometric reconstruction inaccuracies by elevating visual errors into 3D space and rectifying the inaccuracies within that domain. Furthermore, it introduces a cleaning approach for point clouds using a blending formulation during optimization.

Next in Chapter 4, based on Franke et al. [2024], we introduce a stable, versatile, and adaptable scene representation for efficient optimization of scene parameters through differentiation, in addition to enabling fast rendering. This is based on a novel trilinear splatting formulation for smooth point rendering. The representation also allows us to strongly reduce the reliance on a neural network for point rendering processing.

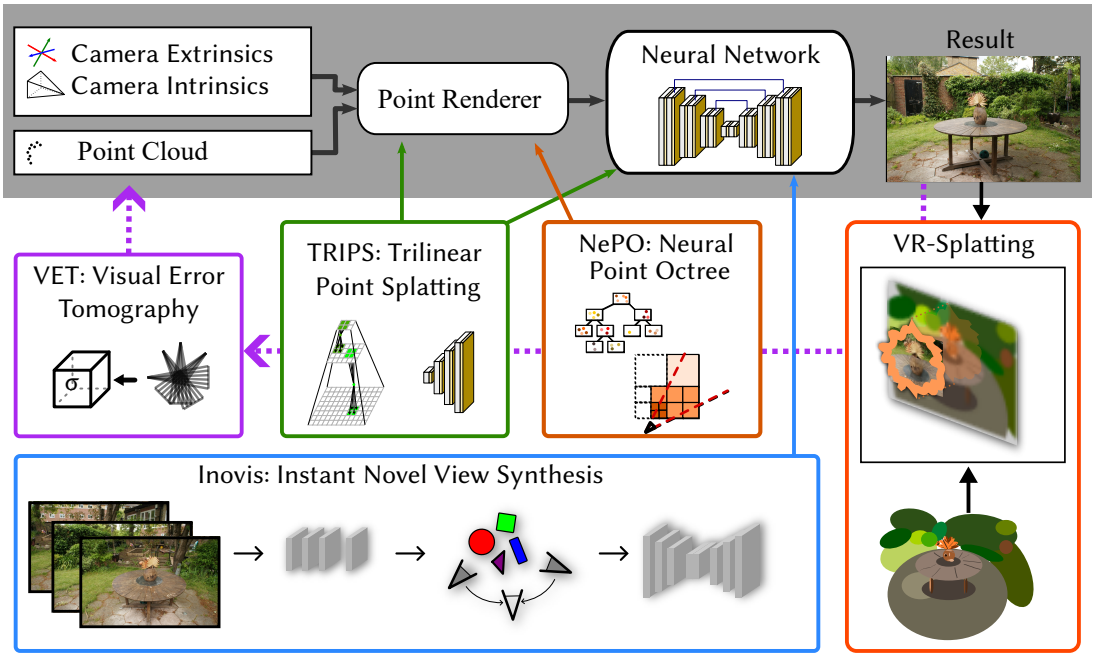


Figure 1.1: Contributions of this thesis. A classical neural point pipeline [Aliev et al. 2020; Rückert et al. 2022] (top, in grey) renders a point cloud with a set of camera parameters, then processes the result with a neural network. We contribute the following to the pipeline: Geometric refinement with VET [Franke et al. 2023] (purple, in Chapter 3). Smooth rendering with stable gradients and smaller neural networks with TRIPS [Franke et al. 2024] (green, Chapter 4), large-scale rendering through hierarchical structuring with NePO [Lewis et al. 2025] (orange, Chapter 5), VR-ready foveated rendering with VR-Splatting [Franke et al. 2025] (red, Chapter 6), and an ad-hoc, zero-preprocessing system with Inovis [Harrer et al. 2023] (blue, Chapter 7).

Subsequently, in Chapter 5, based on Lewis et al. [2025], we expand scalability by accommodating entire city-scale scenes through hierarchical structuring, moving beyond the typical focus on individual objects or room-centric scenes. We group points into an octree structure, and use Level of Detail (LoD) culling strategies to allow for hundreds of millions of points to be processed.

In Chapter 6, we address the challenging hardware requirements of VR rendering by exploiting the visual acuity falloff of the human visual system. This enables us to seamlessly integrate rendering shortcuts that remain visually undetectable. We achieve this by blending the advantages of neural point rendering with Gaussian splat-based rendering [Kerbl et al. 2023], which offer complementary benefits in terms of performance scaling. This chapter is based on the publication Franke et al. [2025].

Lastly, we present a module designed for instantaneous virtual teleportation, eliminating the processing time usually required after scene capture (Chapter 7). This chapter, based on the publication Harrer and Franke et al. [2023], introduces ad-hoc NVS by combining a neural rendering structure from real-time super-sampling with neural point-based rendering and image-based warping.

Our contributions to the neural point-based pipeline are shown in Fig. 1.1. VET (Chapter 3, purple) augments the neural point rendering pipeline by completing the point cloud based on rendering results. TRIPS (Chapter 4, green) and NePO (Chapter 5, orange) change the rendering process with smooth blending or octree-based culling, while using TRIPS also allows shrinking the neural network. VR-Splatting (Chapter 6, red) employs neural point rendering for the fovea and Gaussian splat rendering for the periphery, smoothly blending the two methodologies. Inovis (Chapter 7, blue) changes the neural network to also incorporate nearby views, which removes the need for per-scene neural point optimization.

CHAPTER 2
Background

In this chapter, we review the common capture and processing pipelines for NVS and identify weak points. NVS here describes the task of generating new views of a scene given a set of input images. These images are either captured individually via a digital camera, or in bulk via video recording or specialized capturing rigs, e.g. a light stage.

Traditionally, novel views are rendered after representing a scene as a 4D light field rendering approach [Gortler et al. 1996; Levoy and Hanrahan 1996], a special case of the plenoptic function, which describes the flow of light in a scene. In the absence of occluders and given a sufficiently high capturing density, this function enables the synthesis of novel views through interpolation between captured images. This approach is only dependent on input views for camera positions on a 2D plane. However, the inclusion of 3D geometric information in view-dependent texturing [Debevec et al. 1996] or with unstructured Lumigraph rendering [Buehler et al. 2001] allows arbitrary capture positions and provides good results by warping captured images on the geometry proxy. This is commonly called Image-based Rendering (IBR) [Debevec et al. 1998; Shum and Kang 2000].

The conventional IBR pipeline requires the following three scene parameters:

1. **Extrinsics Parameters (Poses):** To determine where each input image was captured, it is necessary to estimate or obtain its relative pose, including position and orientation. This information is crucial for identifying which image corresponds to specific sections of the scene and, consequently, the novel view.

2. **Intrinsic Parameters (Camera Parameters):** Intrinsic parameters are required for every capture device. In the case of RGB captures, this involves the focal length, optical center, and lens distortions. In addition to poses, these parameters are essential for converting image pixels into 3D space coordinates.

3. **3D data:** A general 3D proxy is required, typically in the form of a mesh. This mesh is derived from a 3D point cloud, created by merging depth maps from individual images. These maps, using combined poses and intrinsic parameters, can be back-projected to form 3D point maps.

A common workflow for IBR involves the use of a photogrammetry software (such as COLMAP [Schönberger and Frahm 2016; Schönberger et al. 2016]) to estimate the necessary scene parameters and use image warping for NVS [Shum et al. 2008] (see Fig. 2.1). However, inaccuracies in pose estimation, intrinsic calibration, or depth reconstruction can introduce errors, leading to misalignments, distortions, or missing details in the final scene representation. These challenges affect the quality and consistency of NVS, highlighting the need for robust reconstruction techniques.

In this chapter, we review the capture process and the existing literature that aims to mitigate reconstruction artifacts. We also discuss weaknesses and benefits, as for our vision, we require a flexible representation, which allows for optimization of all input parameters while allowing fast and perceptually accurate rendering.

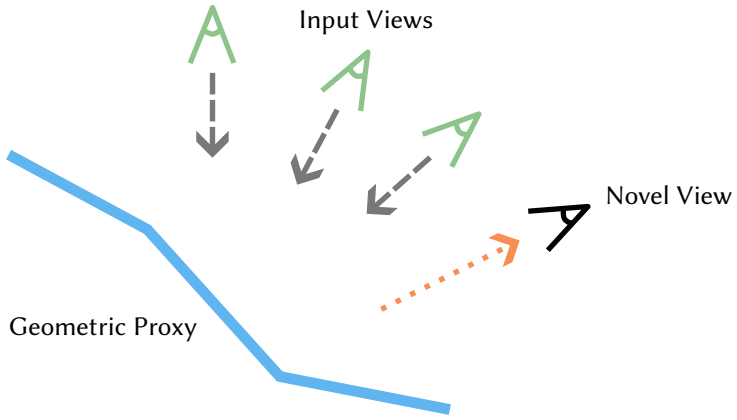


Figure 2.1: IBR scheme. The image data from a set of input views (green) are projected (grey) via a geometric proxy (blue) to a novel view (black).

2.1 3D Proxy and Camera Parameter Estimation

In the following, we recap the common 3D reconstruction pipeline, followed by how active sensing and on-the-fly algorithms fit into this pipeline.

2.1.1 3D Reconstruction (Photogrammetry)

The typical 3D reconstruction process involves Structure-from-Motion (SfM) and Multi-view Stereo (MVS). Initially, the focus is on accurately calculating extrinsic and intrinsic parameters through sparse 3D point triangulation. Subsequently, a detailed 3D model of the scene is generated as a point cloud or triangle mesh.

Structure-from-Motion. In SfM [Snavely et al. 2006], a *sparse reconstruction* is computed, which consists of a couple of thousand 3D points and poses for the input images. The process begins by detecting key feature points in each image using algorithms such as SIFT [Lowe 1999]. These features, which represent distinct parts of the scene, such as edges or corners, are then tracked across multiple images. Feature points are matched between all images and each camera pose is estimated with the feature points, resulting in 3D points, camera poses, and intrinsic parameters. Subsequently, bundle adjustment is used to refine these parameters, iteratively reducing the reprojection errors for the feature points across all images. Using an iterative approach, this process starts with a pair of views and then is extended to include more views [Schönberger and Frahm 2016]. A result of this pipeline can be seen in Fig. 2.2.

An alternative to the iterative approach is the global SfM approach, where the parameters of all images are solved simultaneously. This is done by minimizing the error in the relative rotations first [Hartley et al. 2013], then estimating the camera positions. Finally, global bundle adjustment refines both camera positions and 3D points. This approach offers better scalability compared to incremental methods, but faces key challenges related to accuracy that need to be carefully resolved [Pan et al. 2024].

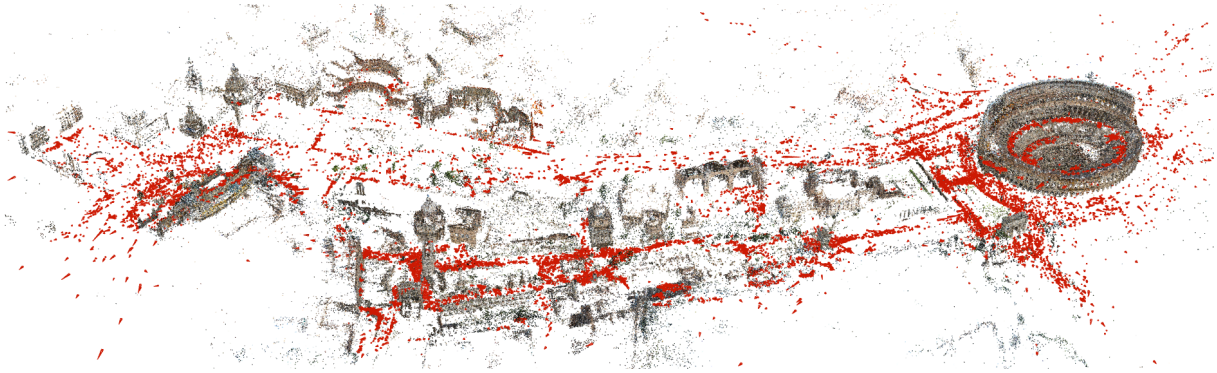


Figure 2.2: COLMAP's sparse reconstruction [Schönberger and Frahm 2016] result, with the cameras colored in red and the sparse point cloud colored with the image colors. The figure is taken from the COLMAP project page [Schönberger 2024], based on Schönberger and Frahm [2016] ©IEEE 2016.

Figure 2.3: Examples of COLMAP's dense reconstruction [Schönberger et al. 2016]. Promising results can be seen, but holes and artifacts are still visible in the point clouds. The figure is taken from the COLMAP project page [Schönberger 2024] based on Schönberger et al. [2016]. Reprinted with permission from Springer Nature Customer Service Centre GmbH: Springer Nature, ECCV, ©2016.

Multi-view Stereo. Unlike the sparse point cloud of SfM, MVS [Furukawa et al. 2010; Goesele et al. 2007; Hartley and Zisserman 2003] aims to create a highly detailed and dense 3D reconstruction. This is done by estimating 2D depth maps for each image, which represent per-pixel distances from the captured position. Estimating depth maps is usually done with matching image patches between images and minimizing dense photometric energy via Normalized Cross-Correlation (as in i.e. COLMAP [Schönberger et al. 2016]). An example of this can be seen in Fig. 2.3. Texture-scarce regions present significant challenges for depth estimation, often resulting in sparse depth maps.

More recently, deep learning-based MVS-net methods [Yao et al. 2018] build on the plane-sweep algorithm [Collins 1996] for reconstruction, sparking strong interest in this subfield. In it, the works improve quality through attention modules [Ding et al. 2022; Liao et al. 2022a; Liu et al. 2023], coarse-to-fine schemes [Cheng et al. 2020; Gu et al. 2020; Wang et al. 2023b; Xu and Tao 2019; Yang et al. 2020c], the use of plane priors have been shown to be effective [Wang et al. 2023b; Xu et al. 2022a; Yuan et al. 2024], recurrent neural networks [Wei et al. 2021; Yan et al. 2020; Yao et al. 2019], 3D Convolutional Neural Networks (CNNs) [Zhu et al. 2021] and vision transformers [Cao et al. 2022, 2024]. These methods dominate the established *Tanks & Temples* benchmark [Knapitsch et al. 2017], however, conventional frameworks such as COLMAP are still more widely used.

The resulting depth maps are then fused, resulting in a *dense 3D point cloud.* Optionally, this point cloud is then meshed (via e.g. Delaunay triangulation).

Discussion. SfM and MVS pipelines offer a flexible and widely used approach for 3D reconstruction, leveraging captured images to estimate camera poses and dense depth information. One of their key advantages is the ability to reconstruct large-scale scenes from standard camera equipment without

requiring specialized sensors. Additionally, these methods benefit from decades of research in feature matching, geometric optimization, and multi-view depth estimation, making them well-supported and robust in many scenarios.

However, the quality of the final 3D reconstruction is highly dependent on capturing conditions and material properties. Pose estimation inaccuracies can introduce misalignments that propagate through the pipeline, leading to depth map inconsistencies that degrade the resulting point cloud or mesh. Similarly, depth estimation errors, particularly in textureless, reflective, or occluded regions, result in missing or noisy geometry. These limitations often cause NVS outputs to suffer from blurring, ghosting, or structural artifacts due to incorrect or incomplete scene information [Chaurasia et al. 2013; Eisemann et al. 2008]. While post-processing techniques can mitigate some of these issues, they add computational complexity, highlighting the need for more robust and adaptive reconstruction methods.

2.1.2 Additional Capturing Modalities

Beyond photo image capture, various additional sensing modalities can enhance the reconstruction process by providing complementary information. These modalities are particularly useful for addressing limitations in depth estimation and camera localization, which are common challenges in purely image-based methods. By incorporating additional data sources, reconstruction pipelines can achieve greater accuracy, robustness, and completeness, especially in challenging environments with textureless surfaces, dynamic lighting, or large-scale scenes. The following sections explore commonly used modalities and their contributions to improving 3D scene reconstruction.

2.1.2.1 3D Data / Depth Capturing

Depth Maps. One form of acquiring depth information is via active depth sensing. This can be captured via structured light [Scharstein and Szeliski 2003; Zanuttigh et al. 2016], whereby a known pattern is cast onto the scene and the resulting projection is interpreted for depth information. This technique is famously used in Microsoft's Kinect. Data from these sensors, however, is often low-resolution and can be noisy [Zollhöfer et al. 2018].

Light Detection and Ranging. Another form of active sensing is LiDAR [Li and Ibanez-Guzman 2020; McManamon 2019; Wandinger 2005]. Thereby, laser beams are cast into the scene and the time the reflected return signal took is measured and converted to distance. This measuring capability yields accurate results for many scenes; however, artifacts are still present when sensing. Some materials do not reflect infrared wavelengths; therefore, no return signal can be measured. Also, thin geometry, such as fences, combined with sensing from a distance can result in laser beams being split; thus part of the signal is returned from the front geometry and part of the background, a process called "mixed pixels" [Tang et al. 2007]. This causes the interpreted distance to be placed in the middle, resulting in outliers or requiring further heuristics, which can cause false positive removal.

To maintain LiDAR data reliability, processing typically involves discarding inaccurate readings to retain only high-confidence data points. Outlier removal methods, such as filtering mixed pixels and weak returns, usually make use of local neighborhood analysis [Tang et al. 2007], but are often specific to vendors and proprietary.

Further Modalities. Recently, further modalities have been considered in the capture process for NVS pipelines. Thermal imaging [Hassan et al. 2025; Lin et al. 2024b; Özer et al. 2025] measures thermal radiation, which is not affected by view-dependent changes, which is advantageous when other modalities show varying signals. Radio Detection and Ranging (radar), which is unaffected by lighting and penetrates small occlusions such as fabric or dust, emits radio waves reflecting off objects. These reflections, captured by receiving antennas, help determine the location and velocities of the objects. Recent interest has grown in combining radar data with depth maps or LiDAR [Domhof et al. 2019; Wirth et al. 2024] to enhance NVS efficiently.

Discussion. While additional capture modalities enhance the quality of the 3D proxy in many scenarios, none provides perfect 3D reconstructions in all use cases. Artifacts and inaccuracies remain, particularly in challenging environments, dynamic scenes, or complex material interactions. Therefore, the methods presented in this thesis aim to rectify erroneous geometry *after capturing*, improving the consistency of the scene and the quality of the rendering.

Throughout the rest of this work, we will be examining datasets acquired through RGB data, depth map streams, and LiDAR. However, we also recognize the potential benefits of integrating other modalities, which could be a fruitful area for further investigation.

2.1.2.2 Active Pose Capturing and Calibration

Independent of the method for 3D data capture, additional modalities can be considered for precise pose and intrinsics. For pose capturing, while Simultaneous Localization and Mapping (SLAM) offers a passive visual approach, active pose capture uses dedicated sensors such as Inertial Measurement Units (IMUs) to track movement and orientation. These techniques provide complementary data, allowing for continuous real-time pose estimation. Furthermore, cameras can be calibrated before capturing to skip estimating the intrinsics of videos.

Simultaneous Localization and Mapping. SLAM is typically used for real-time mapping by a moving device (the agent), with the goal of computing poses and tracking sparse feature points in the environment [Bailey and Durrant-Whyte 2006]. Similarly to SfM, SLAM begins by detecting landmarks or features from captured sensor data and then estimating their relative positions as the agent moves [Bloesch et al. 2015; Jones and Soatto 2011]. SLAM algorithms continuously update the map and refine its location estimate using techniques such as pose graph optimization and probabilistic methods such as the extended Kalman filter [Smith et al. 1990].

A critical component of SLAM is loop closure, which involves recognizing previously visited locations to correct for drift in both the map and the agent's position. By aligning new observations with past data, loop closure improves the overall map accuracy, making it especially important in large or complex environments.

A special case of SLAM are LiDAR-SLAM methods, which work with the LiDAR data for positioning. Generally, they provide high global consistency and precision, mainly due to the 360-degree LiDAR field of view [Hong and Huang 2023].

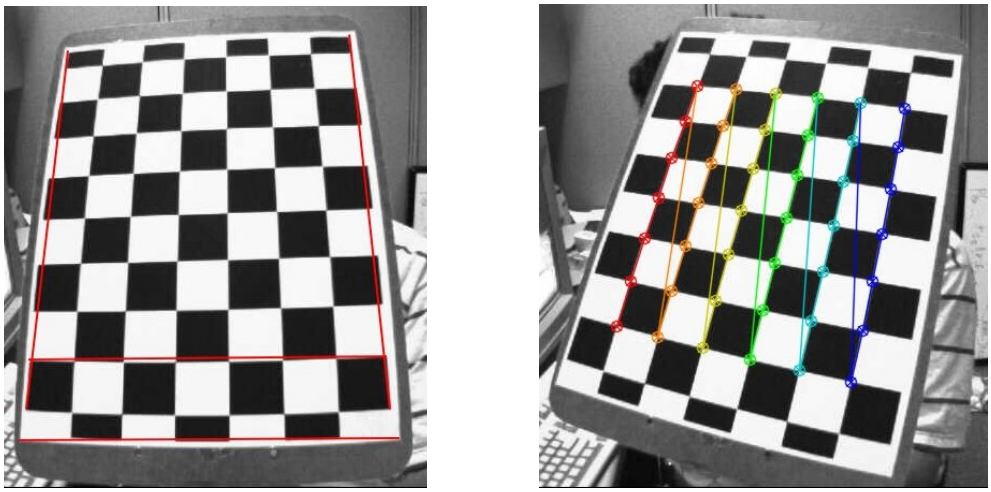


Figure 2.4: A known checkerboard pattern is displayed to a camera (left). Using specific known feature points, this can be used to calibrate camera intrinsics (right). Figure source: OpenCV project page [OpenCV 2025].

Inertial Measurement Units. IMUs are active sensors designed to track movement and orientation, providing pose estimates without relying on visual input. They achieve this by measuring specific force, angular rate, and, in some cases, magnetic fields [Woodman 2007]. Typically, an IMU combines an accelerometer to capture linear acceleration and a gyroscope to measure angular velocity. Although IMUs allow for real-time navigation, they are prone to drift accumulation due to sensor biases, which can lead to inaccuracies over time, especially in the absence of external reference points. To combat this, IMU data are often fused with other sensors, such as GPS or cameras, enhancing navigation and positioning accuracy. This sensor fusion improves overall pose estimation by compensating for drift and increasing reliability in dynamic environments.

Intrinsics Calibration. Rather than estimating camera intrinsics from image sequences, cameras can be calibrated prior to capturing, ensuring precise parameters for the reconstruction process. This typically involves capturing images of a known pattern (see Fig. 2.4), such as a checkerboard, and analyzing how the pattern is projected in the images to calculate the intrinsic parameters of the camera [Heikkila and Silvén 1997]. Calibration also addresses lens distortions, which, if not corrected, can lead to significant measurement errors in the reconstructed scene.

However, calibration is not a one-time task. Various factors such as lens wear, temperature changes, or physical impacts can lead to calibration drift over time. Additionally, in dynamic environments or applications where the camera is subject to vibrations or mechanical stress, misalignments can occur that degrade the accuracy of the calibration. As a result, periodic calibration is necessary to maintain the accuracy of the intrinsic camera parameters [Salvi et al. 2002; Zhang 2000], ensuring consistent performance throughout the capture process.

Discussion. Active pose tracking can be particularly beneficial in scenarios where feature tracking is unreliable, such as in low-light conditions, textureless environments, or scenes with repetitive patterns that challenge traditional visual-based methods. However, these methods are susceptible to drift or other accumulated inaccuracies, and none provide perfect pose estimates over time. Similarly, with intrinsics calibration, it is difficult to guarantee the accuracy of intrinsics without rigorous calibrations, which can be costly. As such, in this thesis, we utilize methods to correct errors related to pose and intrinsics after the data has been captured.

2.2 Image-based Rendering

Following 3D reconstruction, either coloring the mesh or using the mesh as a proxy for IBR can be used for NVS [Shum et al. 2008].

In the area of nearby-view interpolation methods, Vedula et al. [2002] reconstruct dynamic, non-rigid scenes by computing voxel-based shape models and estimating scene flow to interpolate both spatially and temporally. Zitnick et al. [2004] extend this to video-based rendering using synchronized multi-view video streams, leveraging stereo-based depth estimation and matting techniques for artifact-free synthesis. Although these methods generate seamless intermediate views, they remain limited to interpolation.

However, achieving geometrically correct warping requires accurate proxy geometry, which, as reviewed before, is frequently unavailable in real-world datasets. This lack of precision leads to visual artifacts, particularly around object silhouettes, resulting in ghosting and blurred edges.

For artifact mitigation, several techniques were introduced: Eisemann et al. [2008] introduce a warping scheme to align projected textures during rendering using optical flow. Similarly, Chaurasia et al. [2011] use a silhouette-based blending method and Arikan et al. [2014, 2015] use a seam-hiding approach for textures.

Furthermore, per-view optimization was presented for artifact mitigation. These include superpixel-based representations [Cayon et al. 2015; Chaurasia et al. 2013], view-specific meshes [Hedman et al. 2016] or uncertainty-aware depth formulations [Penner and Zhang 2017].

Neural Image-based Rendering Methods. With the advancement of deep learning [Tewari et al. 2020, 2022], several methods integrated deep learning in IBR. Building on conventional plane-sweep volumes, Flynn et al. [2016] introduced using a deep CNN for NVS, which allows for color and depth estimation. This network architecture essentially works as a learned selector and interpolator. Zhou et al. [2016] instead use a CNN to predict the warping operation per pixel, which they call ”appearance flow”.

Hedman et al. [2018] propose using a versatile network predicting blending weights for small mosaics of all warped images. Thies et al. [2020] introduce a neural rendering approach that combines image-based warping with CNN-based blending, by decomposing view-independent and view-dependent effects. Their approach thus also allows for the estimation of lighting changes.

Riegler and Koltun [2020] present a more general system, which renders a proxy depth map for warping and, unlike Hedman et al. [2018] processes an arbitrary amount of input images. This idea is extended by the authors with *Stable View Synthesis* [Riegler and Koltun 2021], which aggregates encoded view-dependent image features on the proxy surface and decodes them after rendering.

Furthermore, Thies et al. [2019] presented the concept of neural textures in the context of IBR, where a high-dimensional appearance texture is optimized jointly with the neural network, allowing it to capture finer details. This approach necessitates scene-specific optimization of both the rendering network and the neural texture, in addition to an inverse rendering process. The method produces impressive results and serves as a foundational technique used in this thesis.

Beyond Triangle Meshes & Including Inverse Rendering. Previous works in IBR often rely on meshed representations of the proxy geometry. While this approach is directly supported by hardware through rasterization pipelines, it presents challenges due to its fully structured nature, making reconstruction and error mitigation more complex.

A step away from triangle meshes are Multiplane Images (MPIs). They are a layered 3D scene representation, consisting of multiple fronto-parallel planes at varying depths, each containing RGB and alpha information, allowing for efficient rendering of new perspectives by compositing these layers. For instance, Zhou et al. [2018] introduced the use of MPIs for view extrapolation from narrow-baseline stereo images, enabling the generation of novel views that extend beyond the original viewpoints. Building upon this, Tucker and Snavely [2020] developed a method to predict MPIs from a single image input, facilitating view synthesis without requiring multiple views. Further advancements include Han et al. [2022], who proposed an adaptive MPI framework to handle complex 3D geometry in diverse scenes, improving the quality of synthesized views. With Local Light Field Fusion, Mildenhall et al. [2019] expanded this concept by expanding views small light fields via MPIs and blending adjacent local light fields. However, in general, MPIs have limited ranges where NVS can be displayed, restricting free exploration [Srinivasan et al. 2019].

Regardless of the representation, there is a need for automatic refinement of the scene parameters based on varying capturing conditions. This refinement can be achieved by integrating inverse rendering techniques into the reconstruction pipeline. Inverse rendering is a long-standing field of computer vision that seeks to recover physical scene properties such as geometry, lighting, and material attributes from 2D images [Goldman et al. 2009; Hernandez et al. 2008; Nam et al. 2018; Xia et al. 2016].

Recent advances in GPU hardware and deep learning have significantly advanced this field, particularly through differentiable rendering techniques. Differentiable rendering aims to enable scene parameters to be optimized using gradient descent-based methods. However, traditional triangle rasterization struggles with analytically correct spatial derivatives due to discontinuities at triangle edges and depth testing [Kato et al. 2020; Loper and Black 2014]. To address these challenges, various methods have been proposed: Genova et al. [2018] and Kato and Harada [2019] introduce gradient approximation techniques, while Rhodin et al. [2015], Chen et al. [2019] and Liu et al. [2019a] employ soft rendering methods, such as alpha blending along object edges, to mitigate these issues. In point-based inverse rendering systems, Wiles et al. [2020] employ softened disk rendering using a Euclidean distance falloff, while Lassner and Zollhofer [2021] propose differentiable sphere rendering to allow for gradient computations. Differentiable rendering is valuable for our purposes as it allows for the improvement of input parameters, in our case, the scene parameters.

2.3 Volumetric and Neural Radiance Field Representations

Automatic scene refinement for IBR requires a differentiable rendering formulation, for which volume rendering is a clear choice, as it is inherently differentiable [Kato et al. 2020] In traditional ray-marched volume rendering, the scene is represented as a density field, and rendering is done with alpha compositing [Porter and Duff 1984]. Specifically, the color C is computed with ray marching N samples:

$$C = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i \qquad \text{with} \qquad T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right), \tag{2.1}$$

where samples of density σ, transmittance T, and color $\mathbf{c}$ are taken along the ray with intervals δ_i.

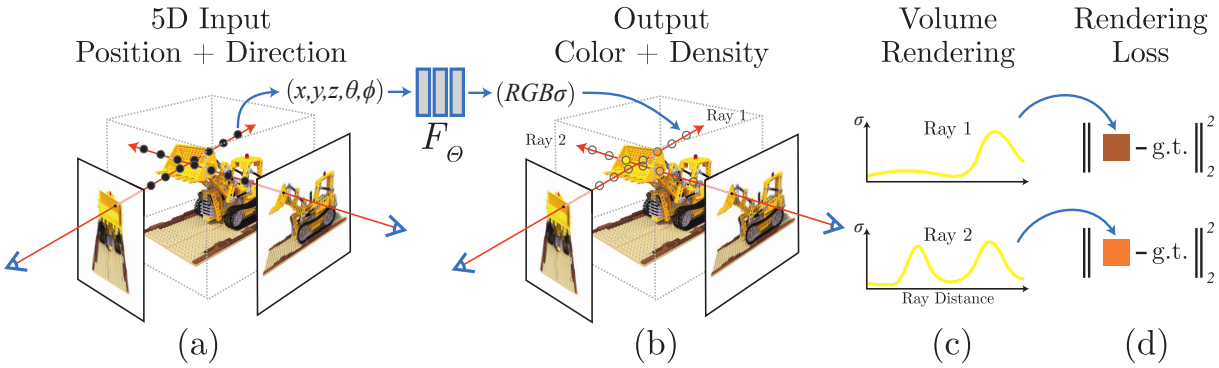


Figure 2.5: The original *Neural Radiance Fields* pipeline. (a) During per-pixel ray marching, 5D coordinates with location and viewing direction are sampled. (b) An MLP transforms the (x, y, z, θ, ϕ) coordinates to (R, G, B, σ). (c) The samples are composited with volume rendering. (d) The weights of the MLP F_θ are updated using the gradients from the L_2 loss of the rendered pixel compared with the ground truth. The figure is taken from Mildenhall et al. [2020]. Reprinted with permission from Springer Nature Customer Service Centre GmbH: Springer Nature, ECCV, ©2020

For IBR, Wang et al. [2021] use this rendering paradigm and introduce an IBR network architecture, which first encodes source images and then predicts colors and densities for each view ray of the target image. They then resolve this via volume rendering, however, without full scene parameter optimization.

Volume rendering can streamline volumetric systems [Henzler et al. 2019; Lombardi et al. 2019; Tulsiani et al. 2017] and voxel grids [Nguyen-Phuoc et al. 2018; Sitzmann et al. 2019; Zhu et al. 2018], and predict signed distance fields for 3D objects [Jiang et al. 2020]. It enables optimizing implicit scene representations via Signed Distance Functions [Liu et al. 2020; Niemeyer et al. 2020; Park et al. 2019; Zakharov et al. 2020] or neural scene functions [Liu et al. 2019b; Mescheder et al. 2019; Niemeyer et al. 2020; Sitzmann et al. 2020], without explicit reconstruction.

The seminal paper here is Mildenhall et al. [2020] introducing Neural Radiance Fields (NeRFs), which represents the entire scene as a Multilayer Perceptron (MLP), which takes the five-dimensional input of (x, y, z, θ, ϕ) (3D position in space and incident view direction) and outputs (R, G, B, σ). Rendering this model is done with ray marching, that is, evaluating this representation multiple times and then compositing the resulting densities and colors. A visualization of this can be seen in Fig. 2.5. This formulation allows the entire scene to be encoded *implicitly*, that is, without *explicit* geometry. Furthermore, the representation is optimized via gradient descent, with supervision via the captured ground truth images. For novel views, arbitrary rays can be defined and sampled. Importantly, to represent fine details, which are normally smoothed by the MLP, positional encodings proved very valuable [Tancik et al. 2020].

The flexibility and quality of NeRF sparked a revolution in implicit scene representations, which are also often called "Neural Fields". Many subsequent works have improved various aspects; however, in the following, we will only discuss a subset of these works.

Initially restricted to bounded scenes, NeRF++ [Zhang et al. 2020] extended NeRF to unbounded scenes, a concept furthered with MipNeRF360 [Barron et al. 2022] using a space contraction towards the scene's center. This allows the MLP capacity to extend further than a predefined $[-1, 1]$ range.

Further improvements regard input view analysis and distributions [Chibane et al. 2021; Kopanas and Drettakis 2023; Wu et al. 2024a; Yu et al. 2021b], scaling [Mi and Xu 2023; Tancik et al. 2022; Turki et al.

2022], anti-aliasing [Barron et al. 2021] and computation times [Barron et al. 2023; Chen et al. 2021; Chibane et al. 2021; Müller et al. 2022; Neff et al. 2021; Tancik et al. 2021]. Effective strategies include scene space discretization through voxel grids [Fridovich-Keil et al. 2022], octrees [Rückert et al. 2022; Yu et al. 2021a], tensor decomposition [Chen et al. 2022], and faster model inference through neurally textured triangle meshes [Chen et al. 2023] or mesh baking [Duckworth et al. 2024; Reiser et al. 2024, 2023; Yariv et al. 2023].

Notably, Müller et al. [2022] gained significant attention with *InstantNGP*, which utilizes hash grids and a highly optimized MLP implementation to achieve faster rendering and training speeds while preserving many of the qualitative benefits of NeRFs.

Regarding quality, *MipNeRF360* [Barron et al. 2022] and *ZipNeRF* [Barron et al. 2023] achieve the highest metric qualities, whereby the latter improves on the 48-hour training times per scene of MipNeRF360 to about 2 hours by combining it with InstantNGP hash grid encodings.

Discussion. Even with these improvements, the NeRF methods are severely limited in rendering performance due to their use of ray marching and multiple evaluations of an implicit structure. They typically render at 10 fps or less. For our vision, we require real-time rendering; as such, we look at explicit representations, which allow fast rasterization capabilities with GPU support [Kerbl et al. 2023].

2.4 Point-based Representations

Explicit representations, such as meshes and point clouds, have the benefit of decades of established rasterization techniques, which accelerate rendering compared to volume rendering. Although triangle meshes were originally used for IBR, the combination of them with inverse rendering to optimize all the parameters of the scene proved difficult. This is mainly due to the structure of meshes as well as discontinuities at the edges of each primitive, making gradient computations for gradient descent difficult. As such, recent point-based rendering schemes for inverse rendering and NVS gained traction. In the following, we first look at point rendering techniques historically and afterwards in the context of NVS.

2.4.1 Point Rendering

Point-based rendering has been a significant focus in computer graphics since the mid-1980s [Levoy and Whitted 1985], with two primary approaches emerging over time [Kobbelt and Botsch 2004]. The first approach involves rendering points as oriented discs, commonly referred to as *splats* or *surfels* [Zwicker et al. 2001]. In this method, each point is associated with a disc whose radius is determined based on the local density of the point cloud. To mitigate visual artifacts between adjacent splats, techniques such as Gaussian alpha masking are employed, and the discs are composited using a normalizing blend function [Alexa et al. 2004; Pfister et al. 2000; Zwicker et al. 2001]. This ensures smooth transitions and a cohesive surface representation. Enhancements in surfel rendering include methods like Auto Splats [Preiner et al. 2012], which dynamically compute the orientation and size of splats to adapt to varying point densities and surface intricacies, thereby improving rendering quality and performance.

The second prominent approach in point-based graphics is *point sample rendering* [Grossman and Dally 1998], where each point is rendered as a single pixel, resulting in a sparse image representation of

the scene. To address the issue of holes or missing data in the rendered image, various hole-filling techniques have been developed. Iterative methods [Rosenthal and Linsen 2008] and pyramid-based approaches [Grossman and Dally 1998; Marroquim et al. 2007; Pintus et al. 2011] reconstruct the final image as a post-processing step, effectively interpolating missing information to produce a continuous surface appearance. To mitigate aliasing artifacts in dynamic scenes, blending points with similar depth values during rendering has been proposed [Botsch et al. 2005; Schütz et al. 2021], enhancing visual coherence. Notably, software-based implementations of point sample rendering have demonstrated superior performance compared to hardware-accelerated methods [Günther et al. 2013; Schütz et al. 2022; Schütz et al. 2021], as software approaches offer greater flexibility and optimization opportunities tailored to specific rendering tasks.

There exists a third domain, which is raycasting point clouds. Hereby the closest points to a ray are searched for and then interpreted for color, geometry or other attributes. In the context of neural rendering, Xu et al. [2022b] use ray marching and resolve features of nearby points for NeRF-like rendering. Ost et al. [2022] aggregate points features with a transformer, and resolves colors with an MLP. Chang et al. [2023] directly computes ray intersections with the underlying implicit surface, while Zhang et al. [2023] leverages point cloud positions to effectively interpolate scene geometry and Ma et al. [2024] rasterize points in a hash table for searching, accelerating the pipeline. These methods however prove too slow for our use-case, as such we investigate point sample rendering and point splat rendering as rasterization techniques further.

2.4.2 Point Sample Rendering for Novel View Synthesis

Early NVS point rendering techniques, such as Hornung and Kobbelt [2009] use a MVS approach to generate colorized point clouds per view and combine them for novel view rendering, but the method relies on well-calibrated input views and struggles with highly sparse datasets.

Neural Methods. Recently, point sample rendering has shown to provide a good base rendering technique for later refinement via deep CNNs. Pittaluga et al. [2019] use the SfM point cloud rendering, appended with SiFT features as input for a CNN, which fills holes in the sparse rendering and decodes color information. However, this approach struggles with the sparse input rendering, causing blurred renderings, suffers from temporal artifacts in motion, and can struggle to resolve correct object occlusions. Similarly, Meshry et al. [2019] builds on the MVS point clouds and converts point renderings, including color, depth, and semantic label, to a more realistic style by augmenting a re-rendering network with an appearance encoding of a captured image. For their pipeline, see Fig. 2.6. This method, nevertheless, also depends on having high-quality segmentation masks and substantial point densities.

A significant step in this domain was the introduction of the optimization of the characteristics per point through Neural Point-based Graphics (NPBG) [Aliev et al. 2020]. In this paradigm, instead of using per-point colors obtained through projecting images on the point cloud, a higher dimensional vector, called *descriptor* or *point feature*, is used per point (Aliev et al. [2020] uses eight floats per point). Similarly to Thies et al. [2019], these features are optimized via gradient descent compared to the input images, allowing a local appearance to be encoded, which is then decoded with CNN. This proved a powerful way of jointly compressing and decompressing local appearance. Additionally, they employ a multi-resolution rendering scheme, and feed each layer into the corresponding level of the U-Net-like [Ronneberger et al. 2015] CNN. This allows the system to resolve occlusion information efficiently, as lower resolution layers occlude points behind them. The pipeline can be seen in Fig. 2.7.

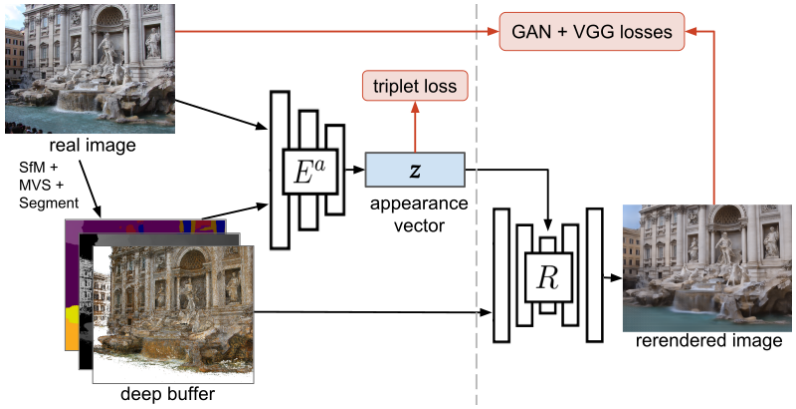


Figure 2.6: The *Neural Rerendering in the Wild* pipeline: A captured real image and corresponding point renderings (including color, depth, and semantic label) are encode with a CNN E^a to an appearance vector z, with which novel views can be rendered in a realistic style using the render network R. The figure is taken from Meshry et al. [2019]. ©IEEE 2019

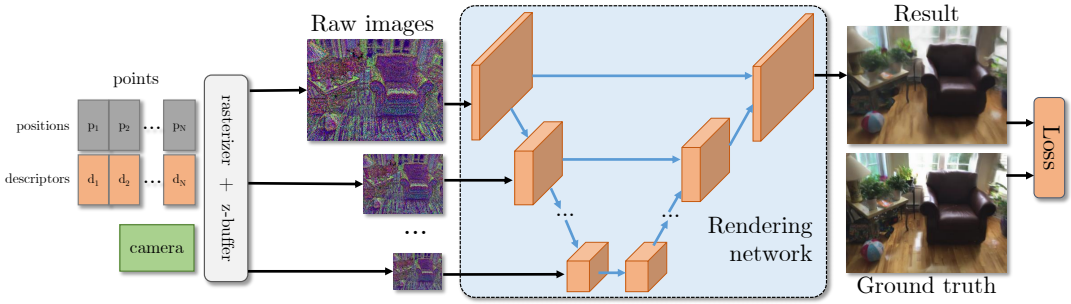


Figure 2.7: The Neural Point-based Graphics pipeline. A point cloud, consisting of points and higher (eight) dimensional feature vector, called descriptor, are rasterized into a multi-dimensional image pyramid (left). Then a U-Net (blue) being fed the resolution layers at each corresponding level resolves the image. Both the descriptors and the U-Net are optimized jointly with gradient decent, using visual loss. The figure is taken from Aliev et al. [2020]. Reprinted with permission from Springer Nature Customer Service Centre GmbH: Springer Nature, ECCV, ©2020

This concept of optimized per-point appearances is also called *neural points*, which will be improved upon in this thesis. For related approaches following NPBG, Dai et al. [2020] show that rendering point clouds into multiple separate layers (similar to MPIs) and optimizing features allow for robust-quality renderings, however, struggles to achieve the same quality as NPBG. Also, building upon the NPBG framework, NPBG++ [Rakhimov et al. 2022] predicts view-dependent neural descriptors for each point in a scene's point cloud through a single *forward* pass of the source images, eliminating the need for per-scene optimization. This approach effectively handles non-Lambertian surfaces through learnable basis functions queried with viewing directions. However, these methods rely on MVS and a point cloud and suffer severely from inaccuracies.

ADOP. In *ADOP: Approximate Differentiable One-Pixel Point Rendering* [Rückert et al. 2022] we introduce differentiable rendering to this pipeline. This paper was co-authored by the author of this thesis; however, it is not part of this doctoral thesis. We augment the NPBG pipeline with two important features: approximate differentiable positional gradients and physically based differentiable sensor

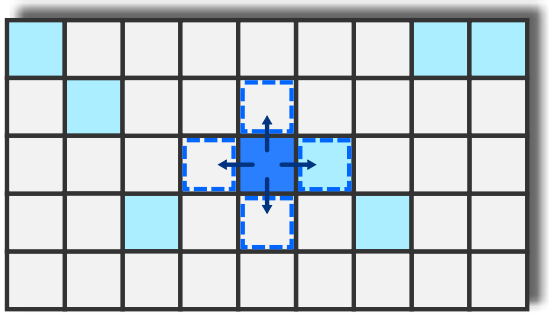


Figure 2.8: Gradient approximation with ADOP: Each point is virtually shifted and the intensity change is recorded as the gradient. The figure is taken from Rückert et al. [2022].

simulation. Previous approaches [Aliev et al. 2020; Rakhimov et al. 2022] only compute gradients with regard to per-point descriptors and the network weights; thus, optimization of scene parameters (pose, intrinsics, and 3D point positions) is not possible. For this, gradients on the image plane (uv-plane) are needed, which are non-trivial due to the rounding operation in the transformation from clip space to discrete pixel space. Here, the approximation involves effectively moving the projected point by one pixel in every direction and calculating the intensity variation *at the target pixels*, assuming the depth value at the target pixel is known. This can be seen in Fig. 2.8. The approximation proves effective for pose and intrinsic optimization, as errors are averaged out over millions of points; however, optimizing point positions showed limited results. The second important part of the pipeline is a differentiable tone-mapping module, which is effective in combating differing exposure levels in images. It also effectively forces feature descriptors to be stored in HDR space. Put together, the pipeline can be seen in Fig. 2.9.

Discussion. While ADOP [Rückert et al. 2022] introduces a fast and versatile NVS pipeline, it falls short of our goals. Firstly, it produces significant aliasing, as points are one-pixel rasterized. Thus small movements cause pixels to jump pixels, causing flickering which is extended by the use of a CNN. Furthermore, while the rasterizer is extremely fast, taking a few milliseconds for millions of points, the CNN heavily impacts performance, especially at larger desired resolutions. Lastly, as gradients are approximate, not all errors in the scene parameters can be fixed with differentiable rendering, impacting quality if, e.g., point clouds are erroneous.

For VR, this means heavily restricting rendering resolution and requiring great capturing of scenes, which might require expansive re-scanning or even manual intervention by a 3D artist.

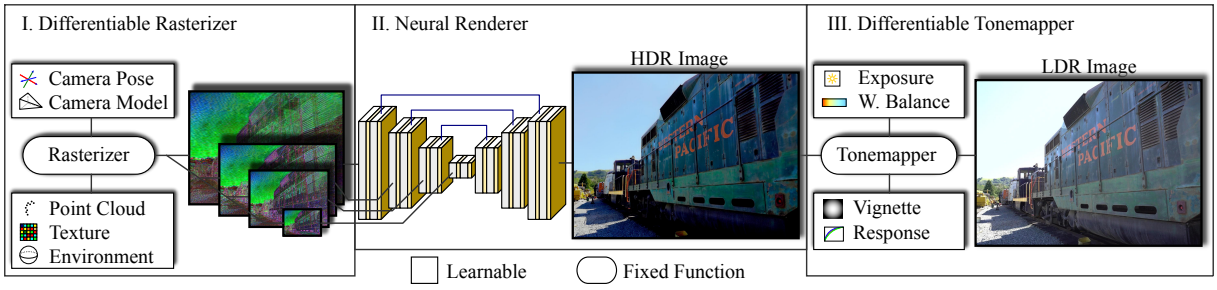


Figure 2.9: The ADOP pipeline: Similar to NPBG, points are rasterized into multilayered feature maps (I.) and resolved by a U-Net (II.). Following is a tone-mapping module (III.), which accounts for exposure difference in captured images. Furthermore, the rasterizer uses approximate spatial gradients, allowing optimization of scene parameters. The figure is taken from Rückert et al. [2022].

2.4.3 Splat Rendering for Novel View Synthesis

As mentioned above, another application of points is in splatted rendering. This approach includes two distinct paradigms: opaque splats and transparent splats.

Opaque splats are rendered as flat discs, often colored or textured, oriented perpendicular to the surface normal, capturing the local surface geometry and appearance. Ellipsoidal surfels extend this concept by modeling each surfel as an ellipsoid, using anisotropic scaling and allowing for better representation of surface curvature and detail [Alexa et al. 2004].

Transparent splats extend traditional surfel rendering by incorporating blending, allowing for the representation of semi-transparent materials and smooth transitions between splats. Instead of rendering opaque discs or ellipsoids, transparent splats use weighted blending or ordered blending functions to accumulate color and transparency information, often leveraging Gaussian blending or order-independent transparency techniques [Alexa et al. 2004].

Differentiable and Neural Splat Rendering. This representation is very flexible and can be enhanced with deep learning and neural rendering. Bui et al. [2018] employ deep neural networks to improve colored surfel rendering, which also efficiently fills holes in between ellipsoid opaque surfels. Similarly, Yang et al. [2020a] use a Generative Adversarial Network (GAN) in the surfel rendering, which is used to create realistic data for autonomous driving.

In the domain of differentiable rendering, Yifan et al. [2019] introduce using Gaussian surfels. Using Elliptical Weighted Average splatting [Heckbert 1989; Zwicker et al. 2001], they are able to propagate gradients across the scene for effective shape optimization.

For NVS, Kopanas et al. [2021] extend this approach for bi-directional warping, which they use for the optimization of per-view features for each captured image. As input, they employ an MVS cloud, and use an IBR approach whereby all scene parameters are optimized based on gradient descent. Furthermore, they introduce using sorted alpha blending, which avoids introducing discontinuities at splat edges. Additionally, they exploit a CNN for neural rendering of the target view.

Expanding on this, Kopanas et al. [2022] introduce an approach primarily for catacaustic rendering. They model reflections as a reflection point cloud, using the surfel formulation and use a neural field for warping these points to represent the reflections. Compared to their previous work, they optimize a global point cloud with high-dimensional features (similar to Thies et al. [2019] and Aliev et al. [2020])), as well as splat sizes and opacities. Furthermore, they employ an MLP for feature decoding, which increases the temporal stability for them.

Zhang et al. [2022] introduce a similar approach using 2D Gaussian discs on sparse point clouds and optimizing the scene representation with images and with videos. They, however, focus on object-centric scenes without backgrounds.

3D Gaussian Splatting. The seminal paper in this domain is *3D Gaussian Splatting for Real-Time Radiance Field Rendering* [Kerbl et al. 2023], often shortened to 3D Gaussian Splatting (3DGS), which introduces anisotropic 3D Gaussians for efficient NVS. The method initializes a sparse set of 3D Gaussians from SfM reconstructions, each parameterized by position, covariance matrix, opacity, and directional color information in the form of spherical harmonics. Through an interleaved optimization process,

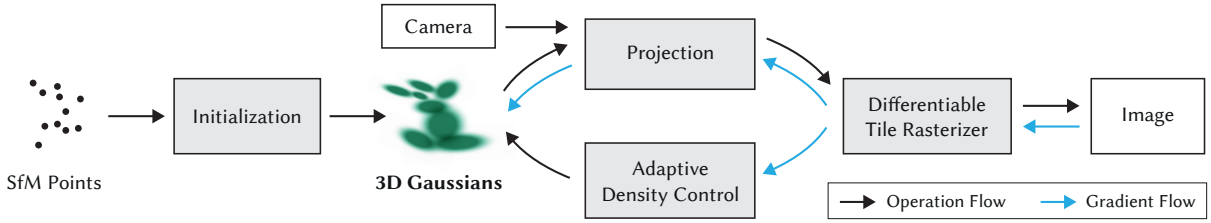


Figure 2.10: The 3D Gaussian splatting pipeline: A sparse SfM point cloud is initialized to anisotropic 3D Gaussians. Differentiable rendering of them optimized the geometry and appearance, while and adaptive density control module uses positional gradient heuristics to add additional primitives. The figure is taken from Kerbl et al. [2023]. Reprinted with permission from ACM, TOG, ©2023

these 3D Gaussians undergo adaptive density control and pruning based on positional gradient heuristics, allowing the scene representation to adapt during training while maintaining geometric accuracy. Unlike NeRF-based approaches or previous neural point renderings, this explicit representation allows direct, real-time rendering at high resolutions without requiring neural network inference per pixel. This is helped by their use of a tiled renderer with approximate, per-primitive sorting, which allows tremendous rendering speeds. Their pipeline is visualized in Fig. 2.10.

The efficiency and explicit nature of this representation sparked a new field, summarized in surveys by Chen and Wang [2024], Dalal et al. [2024], Fei et al. [2024] and Wu et al. [2024b].

Several works have proposed algorithmic extensions, such as Yu et al. [2024] introducing Mip-splatting for anti-aliasing, Huang et al. [2024b] developing 2D Gaussian Splatting for improved geometric (mesh) reconstructions, and Kheradmand et al. [2024] refining the adaptive density module with a space-exploration and sampling scheme. Radl et al. [2024] addressed the approximate sorting of Gaussians with per-pixel sorting and Mallick et al. [2024] enhanced performance, lowered memory requirements, and increased quality.

Large-scale variants have also been explored, with Ren et al. [2025] introducing Octree-GS for hierarchical LoD rendering and Kerbl et al. [2024] optimizing scalability for large-scale scenes through divide-and-conquer. In robotics, Meyer et al. [2024] leveraged 3DGS for dataset generation by merging environment and object scans, simulating realistic placements with a physics engine. Dynamic content has been another focus, with Luiten et al. [2024] integrating temporal coherence for moving objects, Yang et al. [2024] introducing deformable Gaussians with a deformation field, and Huang et al. [2024a] exploring user-controlled motion editing.

Discussion 3DGS-based methods prove very effective and efficient for rendering NVS. However, directly employing 3DGS for our use case is not possible. Firstly, apart from Gaussian position optimization, other scene parameter errors are not optimized for. Furthermore, primitive reconstruction via gradient heuristics can miss details. Lastly, performance and quality for 3DGS heavily relies on the number of primitives used, as seen in Fig. 2.11.

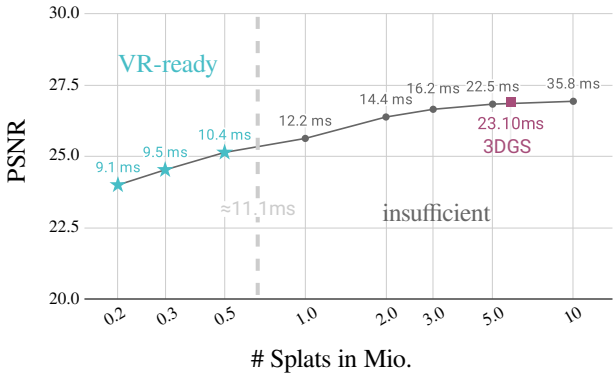


Figure 2.11: Primitive scaling graph for 3DGS. With the driver recommended settings for the HTC Vive Pro [HTC 2019], the number of Gaussians need to be kept low, impacting quality (GARDEN scene, half res. dataset [Barron et al. 2022]).

2.5 Outlook for this Thesis

NVS has evolved from traditional IBR techniques to more advanced methods that take advantage of 3D scene representation optimization and neural networks. Techniques like NeRF have revolutionized the field by enabling novel high-quality views through differentiable rendering and implicit representations.

Alternatively, point-based rendering methods, such as 3DGS and ADOP, offer a flexible and efficient rendering alternative. However, artifacts are still present in reconstructions, mainly due to inaccuracies in intermediate scene parameter estimations, directly impacting the quality of NVS.

To address these challenges, this thesis introduces methods for high-fidelity VR-ready NVS by efficiently refining scene parameters and correcting geometric reconstruction inaccuracies. Building on point-based representations, we further enhance scalability, optimize rendering for VRhardware, and enable instant virtual teleportation through advanced hierarchical structuring and adaptive scene optimization. The contributions of this thesis and improvements to the neural point rendering pipeline can be found in Chapter 1 and Fig. 1.1.

CHAPTER 3

VET: Visual Error Tomography for Point Cloud Completion and High-Quality Neural Rendering

This chapter is based on the publication Franke et al. [2023]. The author of this thesis contributed the algorithmic idea and its implementation. In addition, he evaluated the proposed method and wrote the original draft of the paper. Darius Rückert contributed implementation and methodology advice, while Laura Fink contributed dataset resources. Matthias Innmann contributed methodology advice and helped with the supplemental videos. Marc Stamminger supervised, administered, and helped conceptualize the project as well as provided critical feedback in all stages. All co-authors contributed review and editing during writing.

3.1 Introduction

Neural point-based methods, e.g. ADOP [Rückert et al. 2022], build on point proxies, that is, point clouds originating from MVS or LiDAR scans. To achieve high render quality, a high-quality proxy is required. Different capture methods may fail in various situations, as previously noted, making availability inconsistent. MVS depth estimation cannot reliably estimate featureless, reflective, or transparent surfaces, and often misses small, fine structures. Active scanners, such as LiDAR, can handle textureless surfaces but produce artifacts on dark and glossy surfaces and generally exhibit lower resolution. As a result, the proxy geometry is often incomplete or contains outliers.

In this chapter, we propose a method to rectify these issues using a pair of *cleaning* and *spawning* steps. The key to our approach is a differentiable point renderer that considers and optimizes point transparencies. Points that receive high transparency are considered outliers and can be removed without impacting quality. This *cleaning* operation enables us to improve the proxy point cloud twofold: first, outliers from the original proxy, as they appear for instance in reflective regions or at object edges, can be removed to clean up the proxy and improve render performance. Second, it opens the door for an effective *spawning* operation: We can tentatively add points in likely undersampled regions and let the cleaning step discard points that do not contribute to better renderings. Thus, the surviving added points improve the proxy, with the quality of this point completion being dependent on the accuracy

Figure 3.1: The initial point cloud of the Cherry Tree lacks completeness, with branches and leaves missing. Neural point-based rendering exhibits severe artifacts in these regions *(left)*. By applying VET and spawning new points, we can complete the point cloud efficiently. This appearance-driven geometric completion improves quality of the neural rendering significantly *(middle)* thus outperforming state of the art methods *(right)*.

of the prediction for undersampled regions. To identify these critical regions in 3D space, we introduce a method that we call Visual Error Tomography (VET). Based on the output of the neural renderer and the corresponding input images, we generate error images and project these back to 3D space using computed tomography. With this step, we can reliably identify 3D regions, where the original 3D reconstruction generated too few points, or no points at all, like at glossy surfaces or thin structures. As we show in our results, this step is much more effective than point-growing approaches [Furukawa et al. 2010; Schönberger et al. 2016], which tend to generate many unnecessary points, require many iterations to fill holes, and fail for thin structures. We also show that by spawning (and cleaning) points in a few layers of nested environment maps, it is possible to faithfully re-render the environment, which improves rendering quality considerably.

The teaser image (Fig. 3.1) shows the point cloud for a tree obtained from MVS reconstruction along with the rendered result on the left. In the center, we present the enhanced point cloud, which notably enhances the rendering quality in comparison to other methods that fail to reconstruct the fine-grain structure of the leafs faithfully. In summary, this chapter contributes.

- A fast differentiable neural point-based rasterizer, capable of efficiently handling and optimizing point transparency.
- The concept of Visual Error Tomography, a novel technique enabling reliable identification of erroneous 3D regions.
- A pipeline incorporating iterative steps of point spawning and cleaning, resulting in the generation of clean and complete proxies, ultimately leading to a significant improvement in rendering quality.
- The faithful visual reconstruction of the environment using nested point-based environment maps.
- An evaluation of these methods, showing improved render quality particularly in difficult regions.

An open source implementation of the method is located at:

`https://github.com/lfranke/VET`

The project page, including video comparisons, can be found at:

`https://lfranke.github.io/vet`

3.2 Related Work

For the purpose of synthesis of high-quality novel views, the method based on NeRF, as exemplified by *Nerf++* [Zhang et al. 2020], *MipNeRF-360* [Barron et al. 2022], and *InstantNGP* [Müller et al. 2022], demonstrate promising results without the need for a proxy point cloud. *Stable View Synthesis* [Riegler and Koltun 2021], using a triangle-based proxy excels through strong neural filtering and pre-training and will be compared against. In the context of point-based graphics, the most relevant prior work in this context is *ADOP* [Rückert et al. 2022], upon which we extend by using a blending approach. A disadvantage of most of these approaches is that they do not have a reliable way to automatically remove and spawn new points. This hinders rendering quality and leads to artifacts if the initial point cloud is erroneous. To overcome this problem, Xu et al. [2022b] have introduced a point growing strategy to insert new points close to existing ones. Their approach demonstrates improved rendering quality

on real datasets. Similarly, Kerbl et al. [2023] split large 3D Gaussians in their pipeline to introduce new points. However, their point-growing strategies may not effectively handle regions where few existing points are available for reference, as well as issues with growth criteria selections [Poux et al. 2022] can be present. To this end, Zuo and Deng [2023] propose an improved spawning technique, which identifies pixels with a large rendering error and randomly spawns new points along the ray passing these pixels. While this approach allows for the reconstruction of missing objects, it is worth mentioning that only a limited number of points are placed on the object's surface. As a consequence, the reconstruction process can require thousands of iterations to complete and is prone to overfitting if points are spawned too close to the cameras.

The core of our method is *visual error tomography* (VET), which is based on *computed tomography* (CT). CT reconstructs 3D volumetric densities from a series of volumetric projections obtained by x-ray imaging [Buzug 2008]. In our pipeline, we use Neural Adaptive Tomography (NeAT) [Rückert et al. 2022] for CT reconstruction because this approach can generate sharp, detailed reconstructions and is able to model a non-linear intensity response of ray integrals, which is required for our non-physical reconstruction task.

3.3 Overview

Fig. 3.2 provides an overview of the pipeline. We start with a set of photographs, their corresponding poses, and an initial point cloud that represents the scene. Each point in the cloud is assigned a four-dimensional feature vector and an alpha value that indicates opacity. A differentiable point rasterizer renders the point cloud to feature images in multiple resolutions, which are then passed through a neural rendering U-Net to generate the final renderings [Aliev et al. 2020]. The rendering loss is used to update point features and opacities via backpropagation. Depending on the initial quality of the reconstruction, optimization is performed on the position of the points and the poses of the camera at this stage [Kopanas et al. 2022; Rückert et al. 2022]. Points with low opacity are identified as outliers and automatically removed during optimization. Further details on the differentiable renderer and the cleaning procedure are provided in Sect. 3.4. The remaining image errors indicate missing geometry in the proxy. To identify these undersampled regions, we apply our novel VET and spawn additional

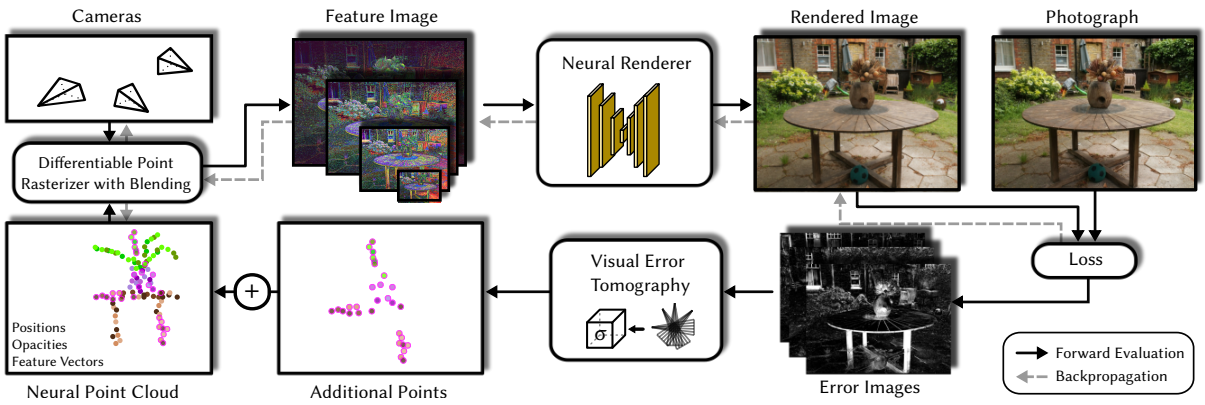


Figure 3.2: Overview of our neural rendering pipeline with visual error tomography. Point descriptors, positions and opacities are optimized via backpropagation, which automatically removes outliers during training. After initial convergence, our VET is used to predict missing 3D points from the error images. Following, the optimization is resumed until the next convergence. This process is repeated 1-3 times depending on scene complexity.

points in the critical regions found. In Sec. 3.5, we provide details about VET and point spawning. The entire process is repeated 1 to 3 times until convergence, depending on scene complexity. As a result, we obtain a high-quality point cloud of the scene, which can be rendered by a real-time neural rendering system in photorealistic quality.

3.4 Point Cloud Optimization and Cleaning

As an initial point cloud, we use the result generated by state-of-the-art 3D reconstruction pipelines such as COLMAP [Schönberger et al. 2016]. In our examples, we use the densified point cloud, but as we show in the evaluation, starting with the sparse point cloud (similar to Kerbl et al. [2023]) and relying on our pipeline to fill in missing points still produces reasonable results.

To better represent the environment, previous methods usually use environment maps [Rückert et al. 2022; Zhang et al. 2020], recently in the form of multi-sphere images [Fridovich-Keil et al. 2022]. In the same spirit, we spawn points on four layers of nested spherical environment maps using spherical Fibonacci sampling [Fink et al. 2019; Keinert et al. 2015] of 0.5M points. These points are optimized and cleaned identically to the original scene points. In particular, with the opacity bias for unseen points (Sec. 3.4.3) this results in sparse and efficient environment maps and high-quality reconstructions of the surrounding.

3.4.1 Differentiable Point Rasterizer with Blending

Our differentiable point-based rasterization approach is based on the work of Rückert et al. [2022]. For detailed information on the forward and backward pass, we refer to the original paper. Our contribution to this pipeline is the use of front-to-back alpha blending instead of the fuzzy depth test. To that end, we first render the points to per-pixel linked lists, sort these lists by each point's depth, and finally blend the colors front-to-back by the respective alpha value. The final color $c_{u,v}$ with alpha blending is defined as

$$c_{u,v} = \sum_{m=1}^{|\lambda|} T_m \cdot \alpha_m \cdot c_m \tag{3.1}$$

where λ is the sorted per-pixel linked list, α_m the alpha value, and c_m the respective color of point m. The transmission variable T_m describes how much light passes through all previous points in the list and therefore ensures that the points at the end of the list only have a small contribution to the final color: $T_m = \prod_{i=1}^{m-1}(1-\alpha_i)$. Since Eq. (3.1) is differentiable, we can integrate this blending approach into the rasterization pipeline of Rückert et al. [2022] to optimize the alpha value of each point. For better optimization, alpha is stored in the range $(-\infty, \infty)$ and only converted to $[0, 1]$ during the blend using the sigmoid function.

The rendering is performed four times with progressively lower resolutions, and these are fed into a multi-scalar U-Net with gated convolutions followed by a neural camera sensor module, as described by Aliev et al. [2020] and Rückert et al. [2022]. Multipixel rendering is efficiently implemented with a custom CUDA per pixel count and collection pass [Selgrad et al. 2015] followed by fast GPU bitonic sorting [Batcher 1968; Franke et al. 2018]. For the backwards pass, only the sorted per-pixel lists have to be stored.

3.4.2 Point Cleaning

The previous optimization adapts the opacity of each point. It is reasonable to assume that the outlier points will receive low or zero opacity. We thus determine outliers by introducing a threshold on the alpha value. We found that during optimization, a threshold of 0.3 works well in all tested scenes and eliminates most outliers, while leaving room for variance during training. For faster training, we remove points below this threshold every 50 epochs.

While previous neural point rendering approaches show some capability for disregarding outliers, mainly through strong masking abilities of gated convolutions, using this explicit optimization of a point confidence proves very powerful in removing outliers, which are either part of the initial reconstruction, environment spheres, or have been added by our spawning step.

3.4.3 Point Opacity Bias

In our point cloud optimization process, we add $\epsilon = 10^{-7}$ to the opacity gradient of each point. This introduces a small bias to the opacity and encourages the system to remove all dispensable points, as unused points will be pushed towards a small opacity value. These points are often environment map points, far away outliers, or inside structures. This bias proves very powerful in reducing point cloud size without impacting rendering quality.

3.4.4 Transition to Opaque Point Rendering

Compared to direct rendering of one-pixel points with depth testing [Schütz et al. 2022], the alpha-based point blending required for outlier removal is slower to evaluate. Therefore, if real-time exploration of the scene is desired, we can smoothly transition our system from alpha blending to opaque depth testing during the optimization stage. To this end, we adjust the sigmoid function, which encodes the raw alpha values α_{raw}, to become steeper over time:

$$\alpha = \text{sigmoid}\left((10 + f \cdot t) \cdot \alpha_{\text{raw}}\right), \tag{3.2}$$

where f is a user-defined parameter and t the current optimization step. After the last iteration, the adjusted sigmoid is almost identical to the step function, with each point having a binary opacity. For the final rendering, we then use only the opaque points without alpha blending, achieving high-performance real-time rendering of point clouds with 100 million points.

3.5 Point Spawning

The core of our point spawning step is visual error tomography (VET, Sec. 3.5.1), which allows us to identify incomplete regions of the scene (see Fig. 3.3), where points then are spawned (Sec. 3.5.2). We apply spawning multiple times when the neural rendering has converged sufficiently, usually every 200 epochs.

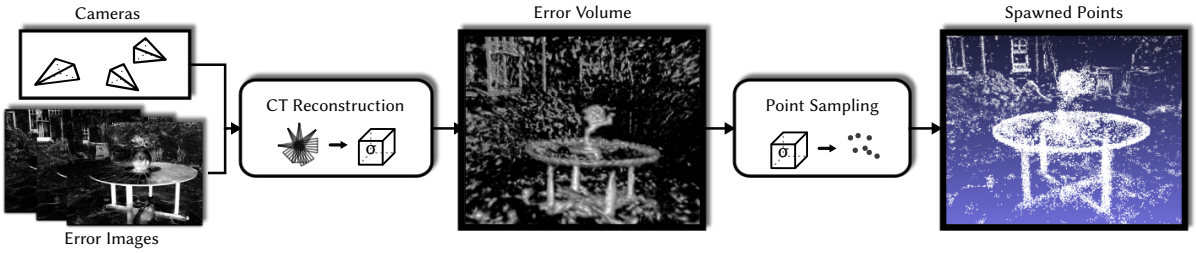


Figure 3.3: Visual Error Tomography: starting from error images *(left)*, we apply computed tomography to reconstruct a 3D error volume grid *(center)*. Within each voxels, random points are spawned, where the number of spawned points depends on the voxel's error value *(right)*.

3.5.1 Visual Error Tomography

The idea of VET is to compute the per-pixel rendering loss of each input image and process these images with a CT reconstruction module. The CT system creates a volumetric model that captures how much each point in space contributes to the visual error. If the CT output for a given 3D point is small, most of the views must have a small visual error at its projected pixel. If the CT output is large, the visual error in the projected pixel coordinates must also be large for multiple views.

Visual errors in neural point rendering often stem from missing points, local minima in descriptor optimization, or strong view dependency. For all three, spawning primitives at that location is a way to improve results. For the first two cases, additional points fill the incomplete geometry or cause the descriptor optimization to take additional training impulses. For strong view dependency, adding points allows the blended descriptors to create a multi-layer surface representation that can approximate view-dependent effects. Although tackling this is not the explicit contribution of our pipeline (unlike e.g. Kopanas et al. [2022]), it does not hinder our VET's purpose.

To compute the error volume, we use the state-of-the-art CT reconstruction approach, Neural Adaptive Tomography (NeAT) [Rückert et al. 2022]. The major advantage of NeAT is its ability to model the nonlinear intensity response of ray integrals, which is an essential property for our case because visual error maps inherently do not follow Beer-Lambert's law of photon attenuation. Furthermore, the neural regularizer of NeAT prefers high-density regions with sharp edges instead of large smooth areas. This has provided better results for our method and produces sharp and clean error volumes, shown, for example, in Fig. 3.3.

For the input images, we smooth the error images I with

$$I'(x, y) = \text{clamp}(I(x, y) * (1 + l) - l, 0, 1) \tag{3.3}$$

to focus on high-error regions and ignore small noise. We use $l = 0.3$ for our error images.

We observe that VET works well over the whole scene, including inside-out capturing scheme and backgrounds, but we limit the volume extent to keep memory costs low, as otherwise we would need to increase the output grid size (see next section). For this bounding box, we use 95% of the inner points of the initial point cloud, as COLMAP or LiDAR commonly have extreme outliers, which are filtered out with this threshold.

3.5.2 Error Volume Point Spawning

The output of VET is a 512^3 voxel grid containing estimated error values. After normalizing this grid to the range $[0, 1]$, we spawn $n_{(x,y,z)} = \lfloor e_i(x, y, z) \cdot p_{\max} \rfloor$ new points, where e_i is the normalized error value and $p_{\max}$ the maximum number per voxel. We use $p_{\max} = 10$, which typically leads to 10k - 2000k newly inserted points per VET step, but this is highly dependent on the error volume. The positions of the n added points are randomly distributed in the cell. It is important to note that adding too many or inaccurately placed points is not an issue due to the robust outlier removal strategy described in the previous chapter.

Theoretically, this process converges, and progressively fewer points are spawned each time, as less and less visual error is represented in the volume. In practice, we spawn points 1-3 times during training, after which no further improvement can be observed. Usually, our spawning and cleaning steps result in point clouds with a similar number of points as the initial reconstruction, and thus have no significant impact on memory consumption and render times.

3.6 Evaluation

3.6.1 Datasets

We have tested our approach on multiple scenes from different datasets: The Mip-NeRF 360 dataset [Barron et al. 2022] has dense point clouds of 5M to 10M points and image resolutions of around 2500×1600 (half the original capture). In the Tanks&Temples dataset [Knapitsch et al. 2017] images were captured with a high-quality RGB camera in full HD and point clouds of 5M to 12M points. The Sydney Opera House scene from Lu et al. [2023] has 2.4M points and an image resolution of 1280×676. The Cherry Tree was captured by us with a point cloud of 2M points and images of 1500×1000. All initial reconstructions were performed using the COLMAP MVS pipeline [Schönberger and Frahm 2016]. Additionally, the Redwood [Choi et al. 2016] dataset is used, which is captured with an RGB-D camera, with small image and point cloud resolutions. For all experiments and methods, unless otherwise noted, 5% of the images (every 20th) were left out of the training and used for evaluation.

3.6.2 Comparison to Related Work

To evaluate the neural rendering performance of our pipeline, we have measured the inference quality on several scenes. We compare against state-of-the-art approaches: Three NeRF-based with InstantNGP [Müller et al. 2022] (with their larger configuration), Mip-NeRF 360 [Barron et al. 2022] and Nerf++ [Zhang et al. 2020], a triangle-based approach in Stable View Synthesis [Riegler and Koltun 2021] and a point-based approach with ADOP [Rückert et al. 2022]. Fig. 3.4 shows a visual comparison on the Tanks&Temples dataset, with the quantitative evaluation presented in Tab. 3.1.

In all scenes, our method outperforms the compared approaches due to our VET-based point-cloud completion. Especially noteworthy is our approach's ability to reconstruct fine detail as well as overall sharpness. In Fig. 3.5, we show four scenes from the Mip-NeRF 360 dataset, observing the same characteristics. Here, however, our PSNR scores tend to be lower than Mip-NeRF 360 because slight errors in calibrations and color shifts by the CNN impact this metric proportionally higher while not impacting perceived visual quality [Zuo and Deng 2023]. However, our rendering times are significantly faster than theirs, as seen in Tab. 3.2 (inference measured on an RTX4090, training on an A100). Each VET reconstruction takes about 15 minutes and is included in our training time.

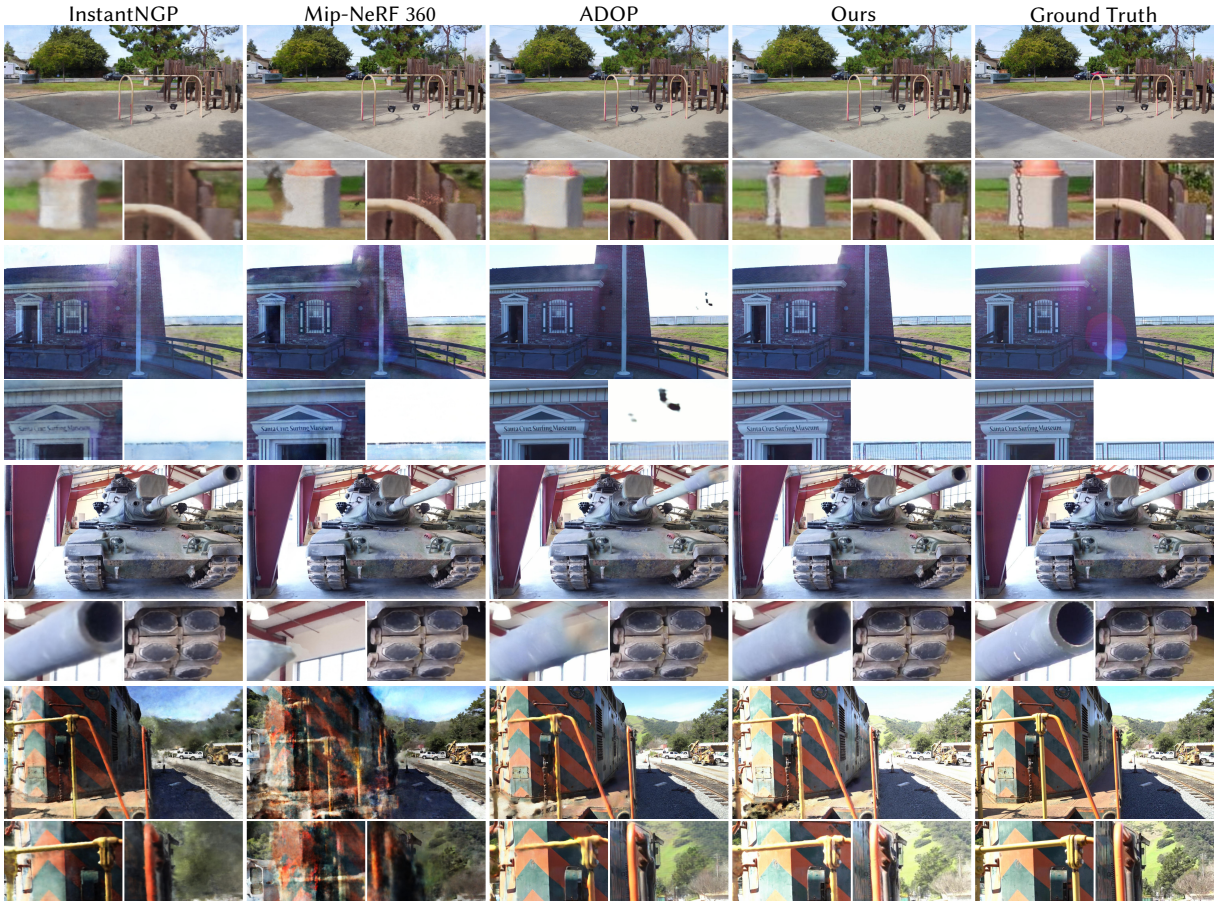


Figure 3.4: Comparison to related work on the Tanks&Temples dataset. Our method is able to reconstruct fine details, remove outliers as well as improving overall sharpness.

Table 3.1: Results on the Tanks&Temples dataset. See also Fig. 3.4 for a visual comparison.

	Train			Playground			M60			Lighthouse		
Method	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑
NeRF++	0.434	18.05	0.579	0.441	22.25	0.613	0.360	23.06	0.728	0.386	20.08	0.662
SVS (half res.)	0.267	17.44	0.683	0.291	22.12	0.704	0.197	23.74	0.831	0.264	17.14	0.722
InstantNGP	0.335	20.46	0.644	0.418	18.69	0.518	0.203	24.97	0.797	0.285	22.98	0.728
Mip-NeRF 360	0.355	18.37	0.624	0.271	24.86	0.736	0.189	24.68	0.838	0.235	21.35	0.750
ADOP	0.131	21.44	0.723	0.126	24.85	0.719	0.103	24.91	0.811	0.119	22.27	0.755
Ours	**0.123**	**21.67**	**0.765**	**0.124**	**25.49**	**0.768**	**0.080**	**27.34**	**0.875**	**0.094**	**24.19**	**0.808**

3.6.3 Ablation Studies

Evaluation of the Error Metric used in VET The VET module takes visual error maps as input to compute the error volume. In Tab. 3.3, we show rendering quality with different VET error metrics on the tree scene (L2 error is processed with $l = 0.01$). Overall, using SSIM provides the best results as it guides more points to be spawned in thin locations.

Narrowing Factor In Sec. 3.4.4, we have introduced a sigmoid narrowing factor to smoothly transition the alpha renderer to traditional depth testing. Tab. 3.4 shows the experiment with different values of f, with large values impacting quality, as changes are too abrupt during training. We therefore recommend $f = 0.01$.

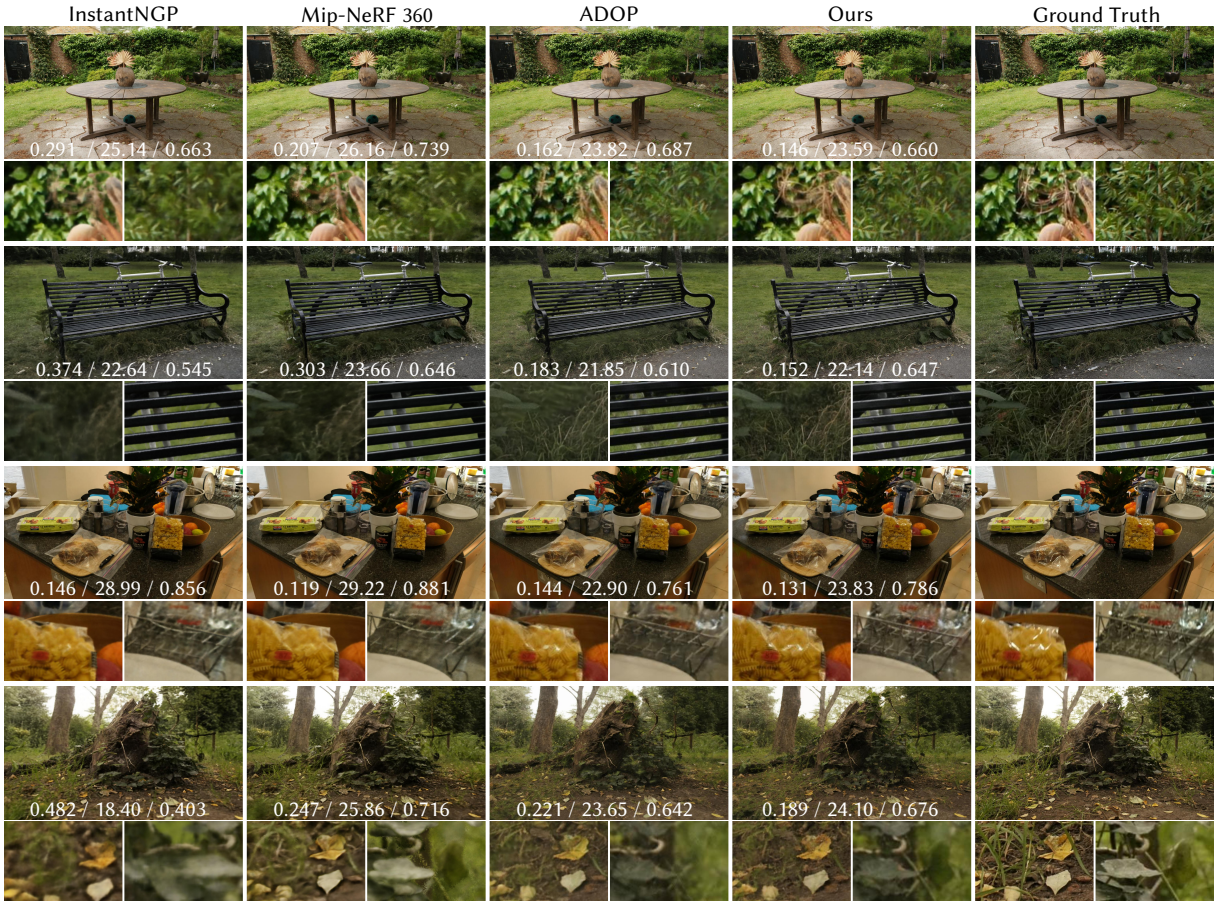


Figure 3.5: Comparison on the MipNeRF-360 scenes with the average LPIPS ↓ / PSNR↑ / SSIM↑ scores denoted.

Table 3.2: Training and render times.

Method	Training	Render (Garden)	Render (Playground)
InstantNGP (8 spp)	0.25 h	1046 ms	1372 ms
Mip-NeRF 360	36 h	38390 ms	18240 ms
ADOP	8 h	30 ms	15 ms
Ours (Blend)	16 h	33 ms	23 ms
Ours (Opaque)	16 h	29 ms	13 ms

Table 3.3: Visual error metrics used for VET on the Cherry Tree scene.

VET-loss	LPIPS ↓	PSNR ↑
use L1	0.177	21.57
use L2	0.187	21.19
use SSIM	0.162	21.88

Table 3.4: Sigmoid narrowing factor f. Transition start after 400 epochs, result after 600 epochs.

f	LPIPS ↓	PSNR ↑
10^{0}	0.200	18.85
10^{-1}	0.197	18.90
10^{-2}	0.186	19.66
10^{-3}	0.191	19.44

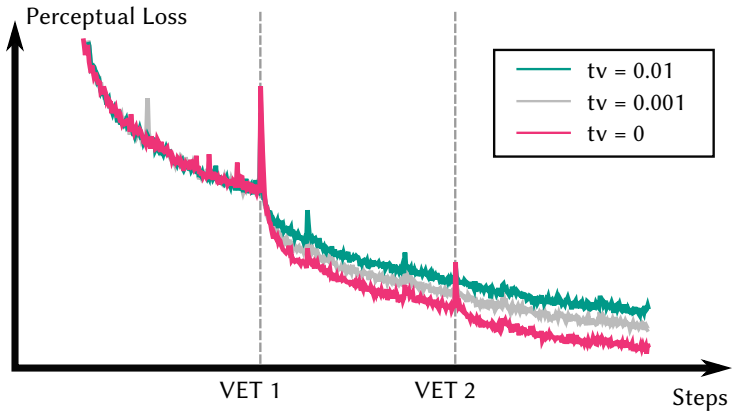


Figure 3.6: Loss graph for running VET with different values of the TV regularizer.

TV Regularization in VET The neural CT reconstruction approach uses a total variation (TV) regularizer to smooth the output volume. We have tested our pipeline with different TV values and found that with a smaller value, more outlier points are spawned. However, since these points are cleaned reliably, the difference in the final result is only marginal (see Fig. 3.6). For the remaining experiments we use $tv = 0.001$ as it is a good compromise between capturing small details and not spawning too many outliers.

Point Cleaning To test the robustness of the outlier cleaning strategy, we have ran our pipeline on the Playground scene, with one million additional random points added. As shown in Fig. 3.7 *(top)* on the right, the final point cloud contains almost no visible outliers. On the Opera scene (Fig. 3.7 *bottom*), this robustness provides great cleanup in outliers placed over the water surface through COLMAP.

Figure 3.7: Ablation study on point cleaning. *(Top)* 1M random points have been added to the Playground scene *(left)* and progressively cleaned *(right)*. On the Opera scene *(bottom)*, COLMAP has placed points originating from the water incorrectly on land *(left)*, which our pipeline can fix *(right)*.

Table 3.5: Point opacity bias (Garden, 7.8M points).

Bias	Points	LPIPS↓	PSNR↑
10^{-5}	0.2M	0.406	20.73
10^{-6}	4.3M	0.186	23.60
10^{-7}	7.3M	0.178	23.70
10^{-8}	8.0M	0.177	23.71
10^{-9}	8.9M	0.178	23.67
10^{-10}	9.6M	0.178	23.67

Table 3.6: Point cleaning threshold (Garden).

T	Points	LPIPS↓	PSNR↑
0.10	10.3M	0.176	23.74
0.20	9.1M	0.177	23.69
0.30	7.3M	0.178	23.70
0.50	6.6M	0.179	23.62
0.70	5.3M	0.183	23.67
0.90	2.6M	0.200	23.44
0.99	0.1M	0.446	19.08

Point Opacity Bias In Sec. 3.4.3, we introduced the opacity bias for the points. As seen in Tab. 3.5, as long as the bias is not chosen extremely strong, low rendering loss is maintained while the amount of points is efficiently lowered. We choose 10^{-7} for our experiments, however we note that 10^{-6} reduces the point cloud efficiently while still providing sensible, if slightly worse results.

Point Cleaning Confidence Threshold In every cleaning step, points with confidence below the threshold are removed. As described in Tab. 3.6, thresholds up to 0.9 still provide reasonable results and reduce point cloud size significantly. However, we use 0.3 for our experiments as it causes point clouds to be similarly sized as the initial COLMAP reconstruction.

Number of Spawned Points per Voxel As described in Sec. 3.5.2 we spawn up to p_{max} points per voxel in the error volume. In Tab. 3.7, the resulting point cloud and the rendering loss is evaluated. We found that $p_{max} = 10$ works well in most cases and maintains the total number of points similar to the initial COLMAP reconstructions. Thus we use this for all experiments. We note however that for difficult cases where COLMAP fails considerably, a high p_{max} can improve quality significantly (e.g. in the cherry tree scene, Fig. 3.1).

Table 3.7: Amount of points spawned per voxel.

	Garden (Init. 7.8M Points)			Cherry Tree (Init. 2M Points)		
p_{max}	# Points	LPIPS ↓	PSNR ↑	# Points	LPIPS ↓	PSNR ↑
2	7.0M	0.181	23.60	1.7M	0.295	18.74
5	7.2M	0.180	23.69	1.9M	0.280	19.39
10	7.3M	0.178	23.70	4.0M	0.223	20.41
20	7.8M	0.177	23.71	17.3M	0.182	21.33
30	8.5M	0.177	23.73	31.5M	0.167	21.88

Environment Map As described, we use four layered point-based spheres for our environment map. Compared with different common environment map strategies (see Tab. 3.8), ours (four point-based spheres) performs best compared to one point-based sphere or to texture-based environment maps

Table 3.8: Environment map setup on the Playground scene.

Method (initially 12.5M Points)	# Points	LPIPS ↓	PSNR ↑
No Environment Map	10.0M	0.146	25.34
1 Textured Sphere	11.4M	0.176	24.99
4 Layered Textured Spheres	11.3M	0.181	25.00
1 Point-based Sphere (init. 0.5M points)	10.8M	0.151	24.59
4 P.-b. Layered Spheres (init. 2.0M points)	13.3M	0.124	25.49

with one [Rückert et al. 2022] or four layers [Yu et al. 2021a]. In contrast to textured maps, our method tends to be less susceptible to overfitting on training view, which in turn improves VET as the error images are more meaningful.

3.6.4 Point Spawning Accuracy

While our method does not explicitly try to increase geometric accuracy of the input model (but tries to minimize visual loss for NVS), we evaluate this accuracy on two training scenes of the popular Tanks&Temples benchmark with their *F-score* based metric [Knapitsch et al. 2017]. As seen in Fig. 3.8, the point cloud is improved by almost 10% compared to COLMAP if the initial dense reconstruction is not cleaned (*ours-full*), thus only VET-spawned points are potentially removed. Otherwise, with our normal pipeline, scores are similar as geometrically accurate points are spawned, but initial correctly placed points might be cleaned if they are not important for visual results of our neural renderer.

	Meetingroom			Barn		
Method	Prec. ↑	Rec. ↑	**F-score** ↑	Prec. ↑	Rec. ↑	**F-score** ↑
COLMAP	0.521	0.247	**0.335**	0.633	0.553	**0.591**
Ours	0.507	0.253	**0.338**	0.631	0.555	**0.591**
Ours-full	0.512	0.283	**0.365**	0.624	0.679	**0.651**

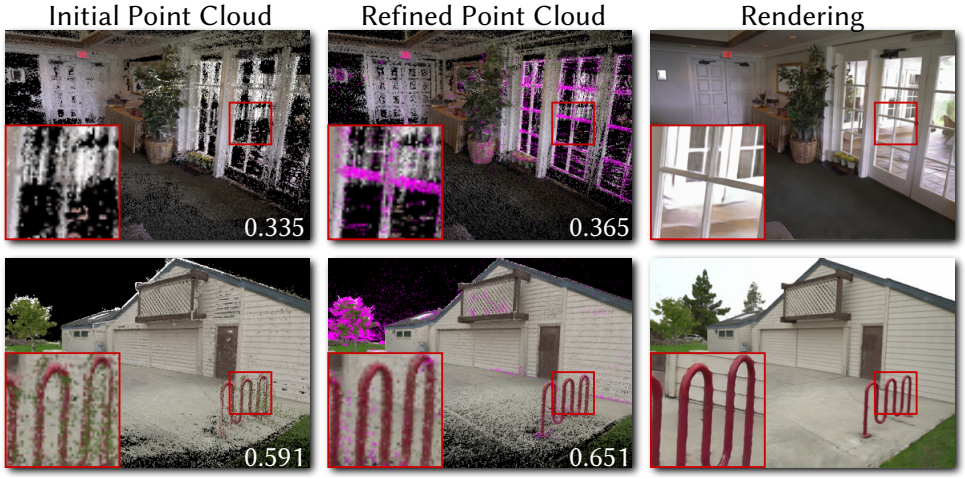


Figure 3.8: Geometric accuracy, measured with the F-score metric [Knapitsch et al. 2017]. Our approach as described (*Ours*) provides similar results in geometry accuracy as COLMAP, while our approach without cleaning the initial point cloud (*Ours-full*) improves scores by about 10%.

3.6.5 Sparse Colmap Reconstruction as Input

We can also skip COLMAP's dense reconstruction and directly work with the sparse reconstruction (similar to Gaussian splatting's pipeline [Kerbl et al. 2023]), thus starting with a point cloud of only 130K points. In this setup, we augment our pipeline with an additional spawning step after 75 epochs, as this low amount of points reaches convergence sooner. As seen in Fig. 3.9, the resulting point cloud and neural rendering provide good overall results.

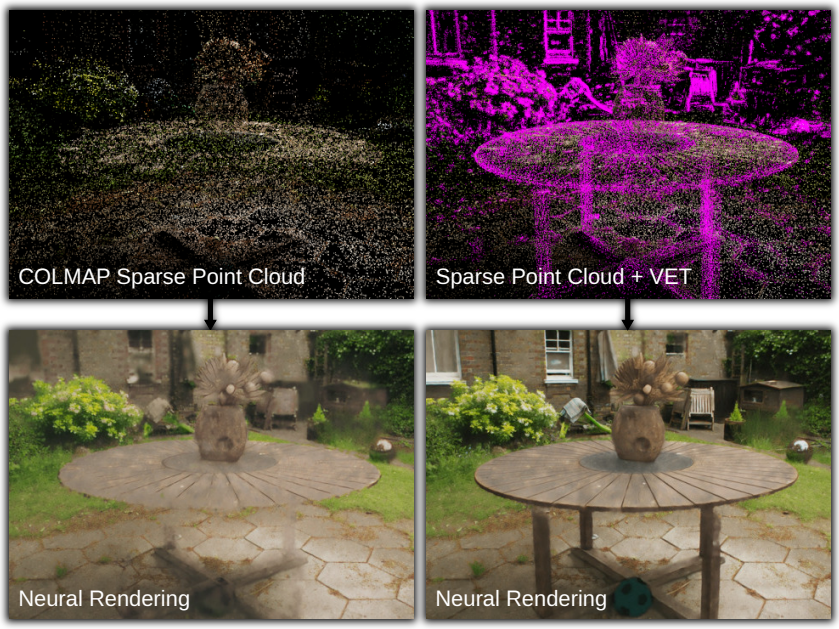


Figure 3.9: COLMAPs sparse reconstruction in our pipeline. Neural rendering without VET *(left)* converges to LPIPS values of 0.596, with VET *(right)* this improves to 0.341.

3.6.6 Different Spawning Strategies

PointNeRF and SNP In Fig. 3.10, we compare to two other NVS methods using point cloud augmentations on their cropped version of the Tanks&Temples Barn with PointNeRF's train/eval split.

PointNeRF [Xu et al. 2022b] spawns points via growing and SNP [Zuo and Deng 2023] samples error map regions in space. However, both fail to reconstruct small details such as the leaves. This is also reflected in the quantitative measurements in Tab. 3.9, where PSNR and SSIM scores are similar, while we outperform both in LPIPS scores.

Table 3.9: Comparison with Point-NeRF [Xu et al. 2022b] and SNP [Zuo and Deng 2023] on their Tanks&Temples dataset.

	Barn (Fig. 3.10)			Avg. Tanks&Temples		
Method	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑
Point-NeRF	0.120	29.15	0.937	0.080	29.61	0.954
SNP	0.109	29.80	0.915	0.079	29.78	0.942
Ours	0.024	29.66	0.939	0.026	29.54	0.950

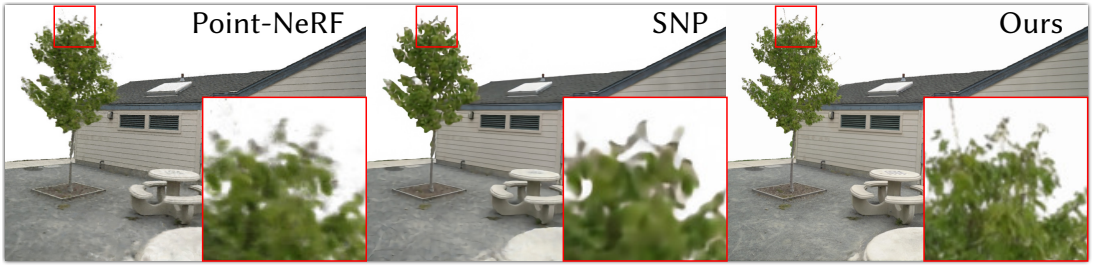


Figure 3.10: Point-NeRF [Xu et al. 2022b] and SNP [Zuo and Deng 2023] use different point spawning strategies with growing and error projection. Both fail to reconstruct fine details.

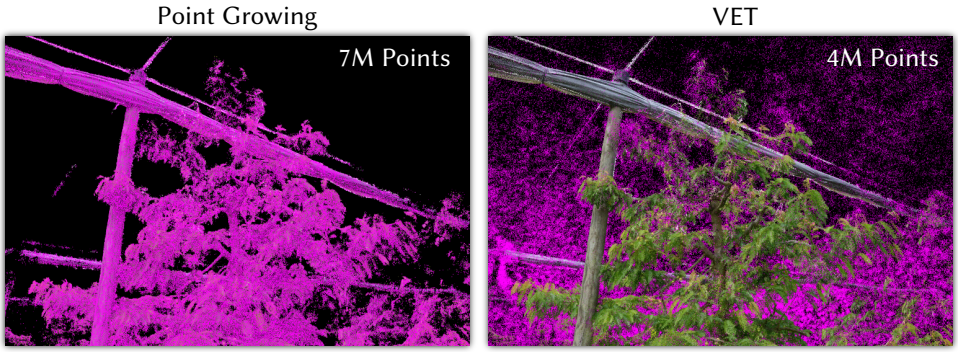


Figure 3.11: Point cloud of the CHERRY TREE enhanced with a point growing strategy *(left)* and our VET *(right)*.

Point Growing in our Framework This result matches our observations. A point growing strategy implemented in our pipeline instead of VET has the same difficulties while increasing point cloud sizes significantly compared to VET, as seen in Fig. 3.11. Furthermore, unprojecting pixel errors and summing them into voxels (similar to SNP's strategy) causes point spawns to be scattered over larger regions. Solving this either requires thresholding which causes details to be missed or requires to use aggressive point cleaning impacting quality. VET, however, avoids all these problems.

3.6.7 View Dependent Effects

While not explicitly modeled in our approach, our blending renderer can handle view dependencies relatively well (as seen in Fig. 3.12) by optimizing reflections into multiple point layers. However, it should be noted that very good angular coverage during training is necessary and explicitly modeled approaches such as Kopanas et al. [2022] may be preferable in this use case.

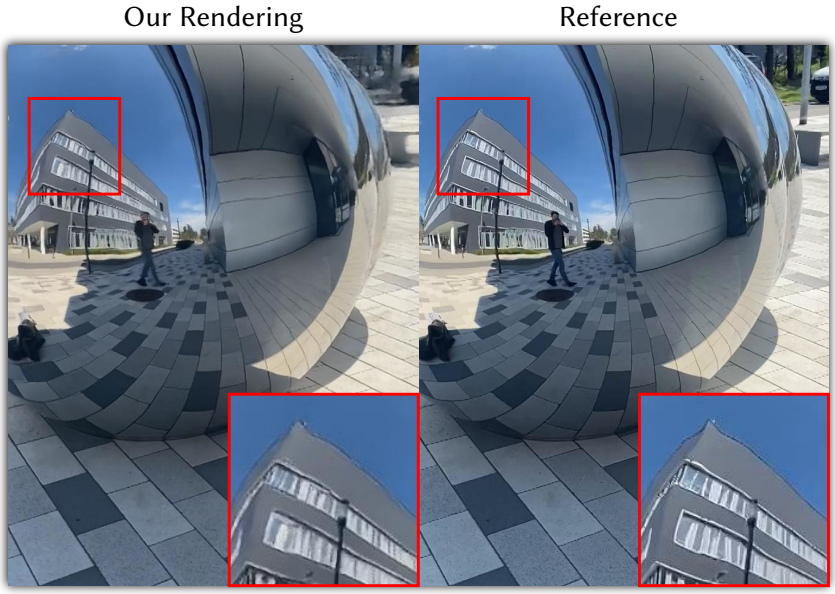


Figure 3.12: Reflecting spheres. Our approach can approximate reflections after point cloud cleaning.

3.7 Limitations

We have encountered a few limitations, inherited from one-pixel point rendering [Aliev et al. 2020], which is efficient, but produces artifacts in rare cases. When the camera gets too close to a surface, the neural rendering network is unable to close the holes between the points, and thus the surface behind the wall becomes visible. Furthermore, we have observed some temporal instabilities when the camera is moved through the scene slowly, which is a side effect of the discrete one-pixel point rendering because points can only discretely move from one pixel to the next.

3.8 Conclusion

To conclude this chapter, we have proposed a point-based neural rendering pipeline that is able to render photo-realistic novel views and can simultaneously refine and complete the input point cloud. Point spawning is implemented using our novel VET technique, which uses visual error maps to identify and populate undersampled regions of the scene. Outlier points are automatically removed by thresholding their opacity value, which is optimized during the training stage. We have shown in several experiments that our pipeline produces high-quality point clouds and outperforms related neural rendering approaches in quality.

CHAPTER 4

TRIPS: Trilinear Point Splatting for Real-Time Radiance Field Rendering

This chapter is based on the publication Franke et al. [2024]. The author of this thesis contributed the algorithmic idea and its implementation. In addition, he evaluated the proposed method and wrote the original draft of the paper. Darius Rückert and Laura Fink contributed implementation and methodology advice, while Laura Fink also contributed to the supplemental video. Marc Stamminger supervised, administered, and helped conceptualize the project as well as provided critical feedback in all stages. All co-authors contributed review and editing during writing.

4.1 Introduction

The results of the previous chapter show that using neural point with VET, we can efficiently complete point clouds. Furthermore, blending points with alpha, which changes the rendering formulation to a formulation very similar to volume rendering [Kerbl et al. 2023] proves efficient for point cloud pruning as well as better optimization. However, there are significant problems with temporal aliasing in this approach through a discrete rounding operation, which also impacts the optimization of structural parameters through approximate gradients.

In this chapter, we introduce TRIPS, a novel trilinear splatting approach that solves these problems. It seeks to harness the strengths of both ADOP (and VET's blending formulation) and 3DGS without losing real-time rendering capabilities, which allow for great optimization, visual quality, and temporal stability.

To recap, 3DGS models each point as a 3D Gaussian distribution, directly optimizing the shapes and sizes of the points. This efficiently fills point cloud gaps in the global coordinate space by employing large splats. In particular, it generates high-quality images without the need for a neural network for

Figure 4.1: Previous point-based radiance field rendering methods provide great results in many cases, but renderings can be aliased and incomplete (ADOP [Rückert et al. 2022] (left), missing parts of the bike's tire), or overblurred (3DGS [Kerbl et al. 2023] (middle), missing fine grass details). Our approach combines the advantages of both to render crisp, complete, and alias-free images.

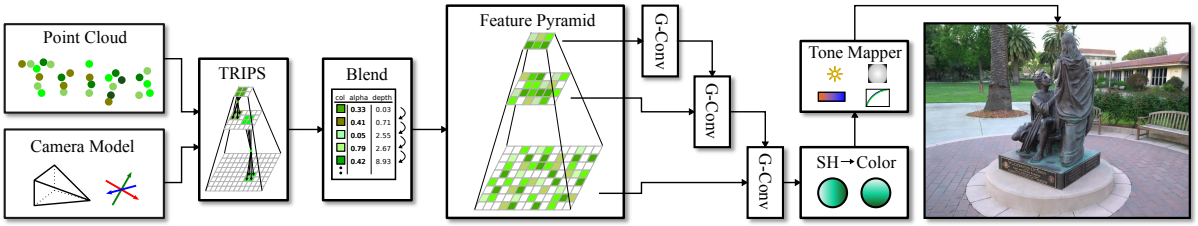


Figure 4.2: Our pipeline: TRIPS renders and blends a point cloud trilinearly as 2×2×2 splats into multi-layered feature maps with the results being passed though our small neural network, containing only a single gated convolution per layer. Following, an optional spherical harmonics module and tone-mapper is used to produce the final image. This pipeline is completely differentiable, so that point descriptors (colors) and positions, as well as camera parameters are optimized via gradient descent.

reconstruction. However, it may reduce sharpness, as Gaussians can cause blurriness and cloudiness, especially with few observations.

In contrast, ADOP [Rückert et al. 2022] rasterizes radiance fields as one-pixel points with depth testing at multiple resolutions. Subsequently, it employs a neural network to address gaps and enhance texture details in screen space. This technique can recreate texture details beyond the original point cloud's resolution, but the computational load increases due to the neural network, and it struggles with large gaps.

Similar to 3DGS, TRIPS rasterizes splats of varying size; however, like ADOP, it also applies a reconstruction network to generate hole-free and crisp images (see Fig. 4.1). More precisely, we first rasterize the point cloud as $2 \times 2 \times 2$ trilinear splats into an image pyramid and blend them using front-to-back alpha blending. Subsequently, we feed the image pyramid through a compact and efficient neural reconstruction network, which harmonizes the various layers, addresses the remaining gaps, and conceals the rendering artifacts. To ensure the preservation of high detail, particularly in challenging input scenarios, we incorporate spherical harmonics and a tone mapping module into our pipeline.

In our evaluations, we show that our approach can produce crisper images compared to 3DGS, with almost the same performance. Furthermore, it surpasses ADOP in the task of filling sizable gaps, allowing correct spatial gradients and maintaining temporal consistency throughout the rendering process. In summary, our contributions in this chapter are as follows.

- The introduction of TRIPS, a novel trilinear point splatting technique for radiance field rendering.
- A differentiable pipeline for optimization of all input parameters, including point positions and sizes, creating robust scene representations.
- An implementation of the method resulting in high-quality real-time renderings under varying capture conditions.

An open source implementation is available under:

`https://github.com/lfranke/TRIPS`

The project page, including videos, is available under:

`https://lfranke.github.io/trips`

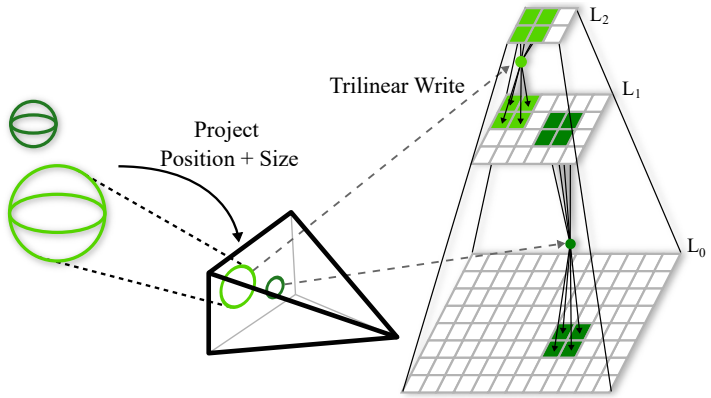


Figure 4.3: Trilinear Point Splatting: (left) all points and their respective size are projected into the target image. Based on this screen space size, each point is written to the correct layer of the image pyramid using a trilinear write (right). Large points are written to layers of lower resolution and therefore cover more space in the final image.

4.2 Method

Fig. 4.2 provides an overview of our rendering pipeline. The input data consists of images with camera parameters and a dense point cloud, which can be obtained through MVS [Schönberger et al. 2016] or LiDAR sensing. To render a specific view, we project the neural color descriptors of each point into an image pyramid using the TRIPS technique (as detailed in Sec. 4.2.1) and blend them (Sec. 4.2.2). Subsequently, a compact neural reconstruction network (described in Sec. 4.2.3) integrates the layered representation, followed by the application of a spherical harmonics module (discussed in Sec. 4.2.4) and a tone mapper that transforms the resulting features into RGB colors.

Core to our method is the trilinear point renderer, which splats points bilinearly onto the screen space position as well as linearly to two resolution layers, determined by the projected point size. Our renderer uses similar nomenclature and is inspired by previous point-rasterizing approaches [Kopanas et al. 2021; Rückert et al. 2022]. The neural image I is the output of the render function Φ,

$$I = \Phi(C, R, t, x, E, s_w, \tau, \alpha), \tag{4.1}$$

where C are the camera intrinsics, (R, t) the extrinsic pose of the target view, x the position of the points, E the optional environment map, s_w the world space size of the points, τ the neural point descriptors, and α the transparency for each point.

In contrast to other approaches, we do not use multiple render passes with progressively smaller resolutions, as this causes severe overdraw in the lower resolution layers. Instead, we compute the two layers which best match the point's projected size and render it only into these layers as 2×2 splat. By doing so, we mimic varying splat sizes, although effectively rendering only 2×2-splats. The layers are then later merged in a small neural reconstruction network (Sec. 4.2.3) to the final image, resembling the decoder part of a U-Net.

4.2.1 Differentiable Trilinear Point Splatting

Using camera intrinsics C and pose (R, t), we project each point position (x_w, y_w, z_w) to continuous (non-rounded) screen space coordinates (x, y, z) and each world-space point size s_w to screen space

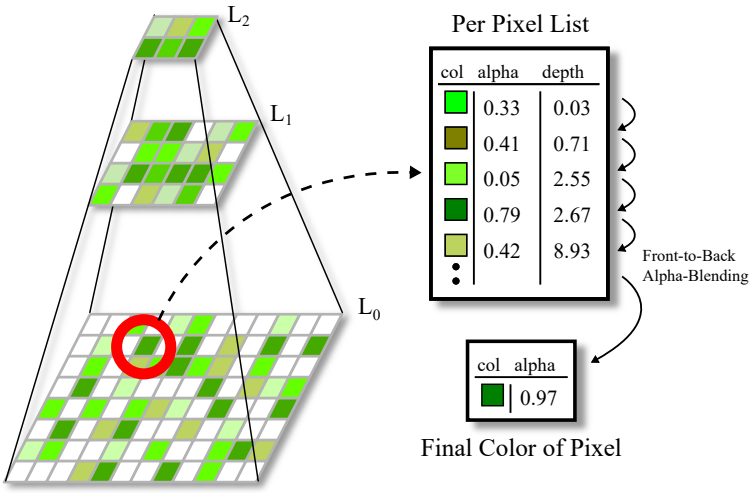


Figure 4.4: In each pixel of the image pyramid, a depth-sorted list of colors and alpha values is stored. The final color of each pixel is computed using front-to-back alpha blending on the sorted list.

size s with the camera's focal length f:

$$s = \frac{f \cdot s_w}{z}. \tag{4.2}$$

Next, we render these points as a $2 \times 2 \times 2$ splats bilinearly and handle point size by splatting into two neighboring resolution layers L, as shown in Fig. 4.3. The resolution layers are selected to be the two closest in sizes to the projected size of the point with $L_{lower} = \lfloor \log(s) \rfloor$ and $L_{upper} = \lceil \log(s) \rceil$.

For each of the then selected eight pixels, we compute the contribution of the point to that pixel and augment its own transparency value with it. The final opacity value γ that is written to the image pyramid for pixel (x_i, y_i, s_i) with $s_i = 2^L$ is

$$\gamma = \beta \cdot \iota \cdot \alpha, \tag{4.3}$$

$$\beta = (1 - |x - x_i|) \cdot (1 - |y - y_i|) \tag{4.4}$$

$$\iota = \begin{cases} 1 - \frac{|s - s_i|}{2^{L_{upper}} - 2^{L_{lower}}} & s \geq 1 \\ \epsilon + (1 - \epsilon)s & s_i = 0 \wedge s < 1 \end{cases} \tag{4.5}$$

where β is the bilinear weight inside the image layer, ι is the linear layer weight, and α the opacity value of the point. The layer weight ι is a standard linear interpolation if the point size s is inside the image pyramid. The second case of Equ. (4.5) handles far away points that have a pixel size smaller than one. In order not to miss these, we always add them to the finest level 0. To avoid that their weight disappears, we ensure that their contribution is at least $\epsilon = 0.25$.

4.2.2 Multi Resolution Alpha Blending

Since each point is written to multiple pixels and multiple points can fall into the same pixel, we collect all fragments in per pixel lists Λ_{l_i, x_i, y_i}. These lists are sorted by depth and clamped to a maximum size of 16 elements. Eventually, the color C_Λ is computed using front-to-back alpha blending (Fig. 4.4):

$$C_\Lambda = \sum_{m=1}^{|\Lambda|} T_m \cdot \alpha_m \cdot c_m \tag{4.6}$$

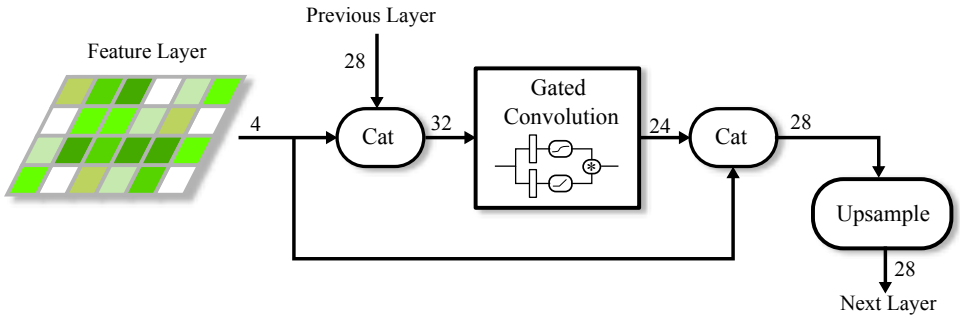


Figure 4.5: Our design of one gated convolution block that processes the features of the image pyramid with the number of channels passed through indicated at each step.

$$T_m = \prod_{i=1}^{m-1} (1 - \alpha_i), \tag{4.7}$$

4.2.3 Neural Network

The result produced by our renderer consists of a feature image pyramid comprising n layers. These individual layers are finally consolidated into a single full-resolution image by a compact neural network, as depicted in Fig. 4.2. Our network architecture incorporates a single gated convolution [Yu et al. 2019] in each layer with a self-bypass connection and a feature size of 32. Additionally, we include a bilinear upsampling operation for all layers except the final one, merging the output with the subsequent level. This configuration is shown in Fig. 4.5 and resembles an efficient decoder network, due to its restrained number of features, pixels, and convolutional operations.

Unlike well-established hole-filling neural networks [Aliev et al. 2020; Rakhimov et al. 2022; Rückert et al. 2022], our approach demands a significantly smaller and more efficient network. This reduced network size stems from the fact that our renderer is adept at filling gaps autonomously and generates smooth output through trilinearly splatting points. Consequently, the network's primary task is to learn minimal hole-filling and outlier removal, allowing it to concentrate its efforts on high-quality texture reconstruction.

4.2.4 Spherical Harmonics Module and Tone Mapping

To model view-dependent effects and camera-specific capturing parameters (like exposure time), we optionally interpret the network output as spherical harmonics (SH) coefficients, convert them to RGB colors, and finally pass the result to a physically based tone mapper. This allows the system to make use of explicit view directions. The SH module makes use of spherical harmonics with degree 2, which corresponds to 27 input coefficients (9 coefficients per color channel). These coefficients are the output of the last convolution of our network. The tone mapper follows the work of Rückert et al. [2022], which models exposure time, white balance, sensor response, and vignetting.

4.2.5 Optimization Strategy

Before novel views can be synthesized, the rendering pipeline is optimized to reproduce the input photographs. This optimization includes point position, size, and features, as well as the camera model

and poses, neural network weights, and tone mapper parameters. We train for 600 epochs, which, depending on scene size, requires 2-4 hours to converge.

As training criterion, we use the VGG-loss [Johnson et al. 2016] which has been shown to provide high-quality results [Rückert et al. 2022]. However, the VGG network tends to be slow to evaluate, thus significantly increasing training times compared to the MSE loss. Therefore, we use a combination of MSE and SSIM [Kerbl et al. 2023] in the first 50 epochs when the advantages of VGG are still negligible. This speeds up training time by about 5% percent.

Similar to Kerbl and Kopanas et al. [2023], we use a "warm-up" period of 20 epochs, during which we train with half image resolutions. Afterwards we randomly zoom in and out each epoch, so that all convolutions (whose weights are not shared) are trained to contribute to the final result.

4.2.6 Implementation Details

Our implementation uses *torch* as auto-differentiable backend, however the trilinear renderer is implemented in custom CUDA kernels, as they commonly provide better performance [Kerbl et al. 2023; Rückert et al. 2022]. Fast spherical harmonics encodings are provided by *tiny-cuda-nn* [Müller 2021].

The renderer is implemented in three stages: collecting, splatting and accumulation, albeit diverging from other state-of-the-art multi-layer blending strategies, this turned out to work best in our scenario [Franke et al. 2018; Lassner and Zollhofer 2021; Vasilakis et al. 2020]. We first project each point (x_w, y_w, z_w) to the desired view and collect each point's (x, y, z) as well as point size s in a buffer, and also count how many elements are mapped to each pixel. This counting is then used for an offset scan to index into one continuous arrays for all layers. The following splatting pass duplicates each point and stores a pair of (z, i) (with i an index to the stored information) in each pixels' list.

Following, a combined sorting and accumulation pass is done. Regarding performance, this part is critical, as such we opt to only use the front most 16 elements from each (sorted) list, a common practice when blending points [Lassner and Zollhofer 2021]. We could not identify any loss of quality caused by this approximation, as the blending contribution of later points is very low. This limitation allows us to use GPU-friendly sorting, as we repeat warp-local (32 threads) and shuffle-based bitonic sorts, always replacing the latter 16 elements with new unsorted ones, until the lists are empty. For the backwards pass, the sorted per-pixel lists are stored, allowing fast backpropagation. The front-to-back alpha blending (see Sec. 4.2.2) is done in the same pass as the sorting pass, because all relevant elements are already in registers.

In contrast to Kerbl and Kopanas et al. [2023], we use this per-pixel sorting, which proved to be faster for us then global sorting. This is mostly due to the higher amount and smaller sizes of points in our approach.

For scenes with a large deviation in point density, we found that occlusion may not be correctly evaluated by the neural network in edge cases. Therefore, we include points from coarser layers during blending (in the usual way), of which the additional cost is very small (< 0.5ms).

Point sizes are initialized with the average distance to the four nearest neighbor, which is then efficiently optimized during training (see evaluation: Fig. 4.7).

4.3 Evaluation

Next, we compare our approach with prior art as well as showcase the effectiveness of our design decisions in ablation studies.

4.3.1 Setup and Datasets

We have evaluated our approach on several scenes from the Tanks&Temples [Knapitsch et al. 2017] and the MipNeRF-360 [Barron et al. 2022] datasets. Additionally, we use the Boat and Office scene from Rückert et al. [2022] to evaluate robustness towards difficult input conditions. The former contains outdoor auto-exposed images while the later is an office floor with multiple distinct rooms and a large LiDAR point cloud, but sparsely placed cameras.

From Tanks&Temples, we use the *intermediate* set containing eight scenes: Train, Playground, M60, Lighthouse, Family, Francis, Horse and Panther. These scenes are outdoor scenes captured under varying lighting conditions but with good spatial coverage and can be seen as a good baseline for robustness. The MipNeRF-360 dataset [Barron et al. 2022] consists of 5 outdoor and 4 indoor scenes. This dataset was captured with controlled setups and has capture positions well suited for volumetric rendering with a hemispherical setup [Kopanas and Drettakis 2023]. We use half resolution for images of this dataset, resulting in resolutions of around 2500×1600 px for outdoor and 1550×1030 px for indoor scenes.

Point clouds of all scenes were acquired via COLMAP's MVS [Schönberger et al. 2016], except Office which was captured by LiDAR.

For the quantitative evaluation, we use the $\text{LPIPS}_{\text{VGG}}$ [Zhang et al. 2018], PSNR, and SSIM metrics. We note, however, that neither of these metrics always reflect visual impression. Some approaches are trained with MSE-loss or SSIM and therefore naturally perform better in PSNR and SSIM. Our approach, on the other hand, is trained with VGG-loss and thus usually shows better scores on LPIPS.

In all experiments, we leave every 8th view out for testing. This is the same train/test split as used in current related work [Barron et al. 2022; Kerbl et al. 2023].

4.3.2 Quality Comparison

In Tab. 4.1 and Fig. 4.6, we compare our approach to InstantNGP [Müller et al. 2022], MipNeRF-360 [Barron et al. 2022], 3DGS [Kerbl et al. 2023] and ADOP [Rückert et al. 2022]. The latter two are the closest-related point-based radiance field rendering approaches.

On the Tanks&Temples dataset, our approach achieves on average the best LPIPS score with an improvement of 20% over the second best. In PSNR and SSIM, the score is on par with state-of-the-art. On the MipNeRF-360 dataset, we obtain again the best LPIPS score; however, the volumetric methods and 3DGS show an improved PSNR and SSIM. The difference can be inspected in Fig. 4.6. For example, in row 3, the TRIPS rendering provides better sharpness with more details, but the MipNeRF-360 and Gaussian output are overall cleaner with less noise. On the difficult boat and office scenes, we can show that our rendering pipeline is robust to extreme input conditions.

Table 4.1: Results on the Tanks&Temples and MipNeRF-360 datasets, as well as Boat and Office. See also Fig. 4.6 for visual comparisons.

	Tanks&Temples			*MipNeRF-360*			Boat			Office		
Method	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑
InstantNGP	0.475	21.74	0.692	0.374	25.94	0.697	0.598	15.34	0.455	0.544	13.45	0.801
Mip-NeRF 360	0.340	24.61	0.789	0.286	**28.23**	**0.796**	0.680	12.20	0.357	0.526	15.22	0.832
Gaussian Spl.	0.300	*24.63*	**0.818**	0.278	*26.94*	*0.792*	0.544	15.30	0.470	0.371	18.77	0.878
ADOP	*0.229*	23.78	0.802	*0.285*	23.26	0.707	**0.301**	**20.49**	**0.650**	*0.279*	**21.47**	**0.899**
Ours	**0.213**	**24.64**	*0.808*	**0.233**	25.94	0.772	**0.301**	*20.38*	*0.633*	**0.271**	*21.36*	*0.887*

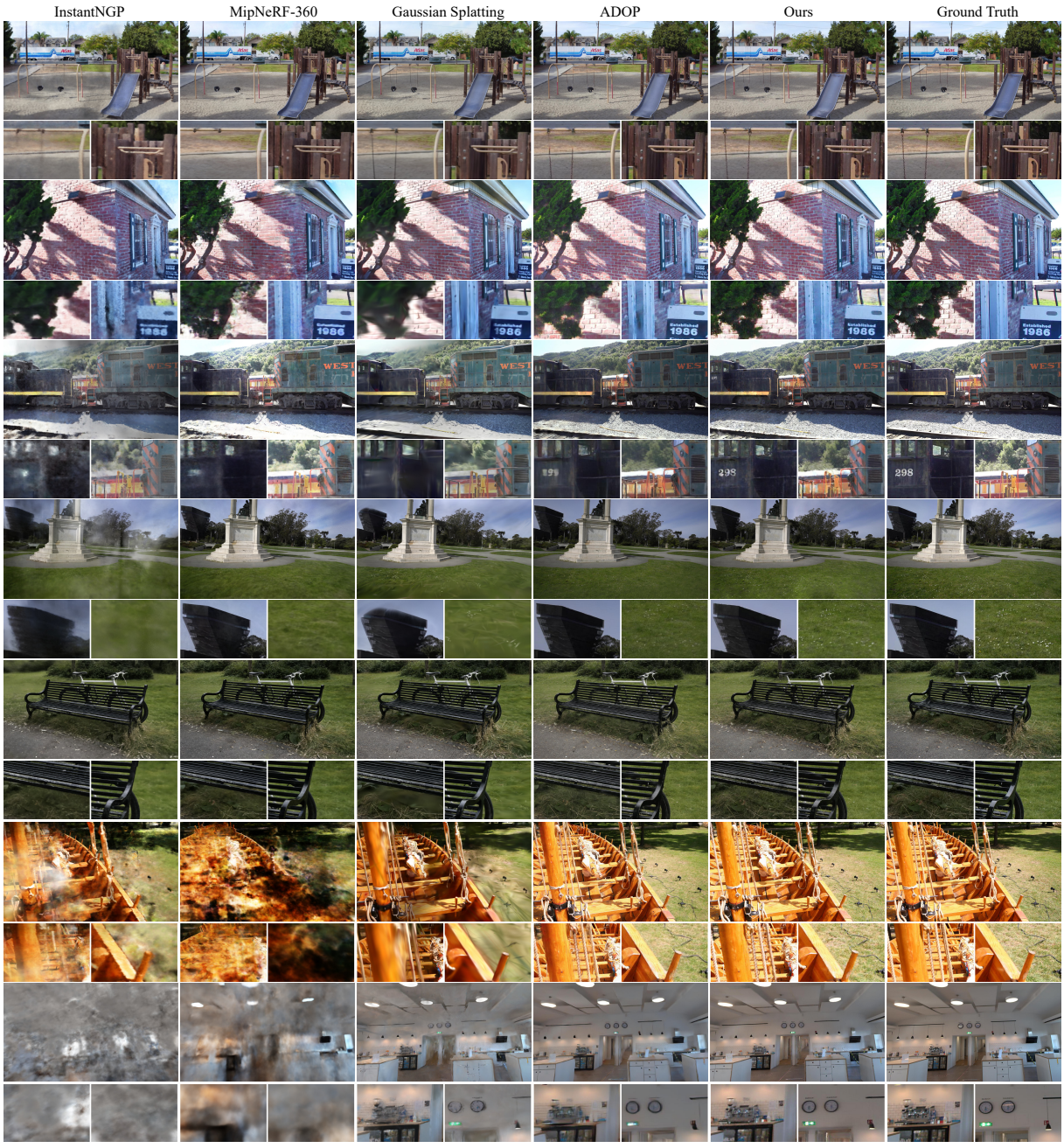


Figure 4.6: Visual comparisons.

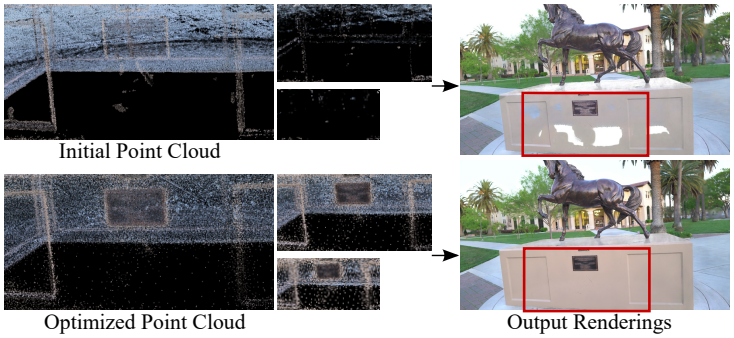


Figure 4.7: The initial COLMAP reconstruction lacks points on the pedestal of the statue (top left). Our approach distributes the few present points and increases their sizes (bottom left) thus rendering them also in lower layers (middle). Thus our pipeline can avoid distracting holes (right).

4.3.3 Ablation Studies

In this section, first we show the effect of our design choices.

Point-Size Optimization With our trilinear splatting technique, point sizes can be optimized to fill large holes in the scene. We show this capability in Fig. 4.7, where the initial point cloud exhibits a large hole in the pedestal of the horse producing artifacts in rendering (top row). To combat this, our pipeline efficiently moves and enlarges the points to fill the hole (bottom row), thus providing great render quality.

Point Position Optimization To test the efficiency of our trilinear point position optimization compared to the (cheaper) approximate gradients from ADOP, we added random noise (of 0.01) to the positions of all points after training, then optimize only point positions for 100 epochs. The result can be seen in Fig. 4.8. Our pipeline is able to reconstruct the correct rendering, while ADOP's result barely improves.

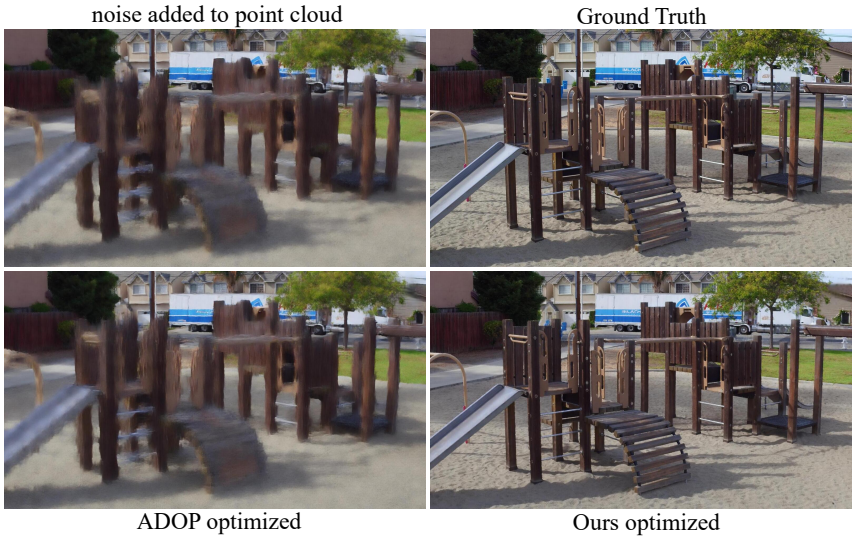


Figure 4.8: We added noise to the converged point clouds of ADOP and ours, then restarted optimization for positions only. Ours is able to converge back to the correct result, ADOP fails at that.

Number of Render Layers Due to our trilinear point rendering algorithm, increasing the number of pyramid layers has almost no negative impact on render time. As seen in Tab. 4.2, having 8 layers improves quality, especially with PSNR. For reference, other approaches make use of 4 [Rückert et al. 2022] or 5 [Aliev et al. 2020] layers and describe significant performance impacts when increasing the number of layers [Rückert et al. 2022].

Table 4.2: Number of resolution layers used (HORSE scene).

#Layers	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓
3	0.216	21.24	0.818	7.10ms
4	0.201	22.40	0.826	7.35ms
5	0.197	22.76	0.826	7.42ms
6	0.196	23.06	0.829	7.50ms
7	0.195	23.10	0.826	7.54ms
8	0.192	23.34	0.828	7.61ms

View Dependency After the neural network, optionally we use a spherical harmonics module to model view-dependent artifacts of the scene. This improves the rendering quality for some scenes (GARDEN), while for others, it makes little to no difference (see Tab. 4.3). Applying the spherical harmonics before the network achieves roughly the same quality, but also reduces efficiency due to additional memory overhead. On scenes without reflective materials, skipping the spherical harmonics module is thus possible.

Table 4.3: View dependency on different scenes. On scenes with strong view dependency (GARDEN), adding view dependent configurations, either via our SH network module (*SH-net*) or optimized per point (*SH-point*) increases quality, however the per-point point setup severely impacts performance. Our module gives a balanced trade off, which also avoids over-fitting on less view-dependent scenes (PLAYGROUND).

	PLAYGROUND (12.5M Points)				HORSE (1.8M Points)				GARDEN (8.2M Points)			
View-dep	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓
none	0.225	24.85	0.720	11.1ms	0.202	22.73	0.822	7.7ms	0.219	24.82	0.756	17.9ms
SH-net	0.225	24.88	0.724	13.3ms	0.203	22.81	0.825	8.9ms	0.222	24.46	0.752	18.5ms
SH-point	0.236	24.32	0.702	27.4ms	0.200	22.89	0.829	10.2ms	0.213	25.15	0.756	27.3ms

Feature Vector Dimensions Our pipeline uses by default four feature descriptors per point. More features only marginally increase the quality, while requiring significantly more memory and slightly increasing rendering time, as shown in Tab. 4.4.

Table 4.4: Features per point on the PLAYGROUND scene.

# Features	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓
4	0.225	24.85	0.720	11.1ms
6	0.231	24.61	0.701	11.7ms
8	0.223	25.04	0.727	12.2ms

Networks In our pipeline, we use a small decoder network made out of gated convolutions, presented in Sec. 4.2.3. ADOP [Rückert et al. 2022] on the other hand, uses a four-layer U-Net with double convolutions for the encoder and decoder (thus around 6 times more parameters). As seen in Tab. 4.5, in our pipeline, our networks provide similar quality to ADOP's full network, while being much faster

in inference. With spherical harmonics, inference times slightly increase, but the system is now able to model view dependency. Adding the SH-module to the second finest layer (ours+SH_{L2}) instead of the finest (ours+SH) of the network improves efficiency but weakens results.

Table 4.5: Network configuration compared (PLAYGROUND scene).

Network	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓
ADOP-net	0.236	24.74	0.713	10.7ms
ours	0.225	24.85	0.720	4.5ms
ours+SH	0.225	24.88	0.724	5.6ms
ours+SH_{L2}	0.248	24.34	0.684	2.2ms

Time Scaling on Number of Points As seen in Tab. 4.6, TRIPS is very efficient in rendering large amounts of points. Even for our largest scene with more than 70M points, the pipeline remains real-time capable with only 15ms required for rasterization.

Table 4.6: Efficiency of our approach regarding point cloud sizes.

Scene	HORSE	GARDEN	PLAYGR.	BOAT	OFFICE
#Points	1.8M	7.8M	12.5M	53.0M	72.5M
Time	2.5ms	5.9ms	6.2ms	13.1ms	15.0ms

4.3.4 Rendering Efficiency

In Tab. 4.7, we evaluate training and rendering time for all examined methods. Our method trains for around 2-4h per scene on an Nvidia A100 and renders a novel view in around 11ms on an RTX4090. A finer breakdown of the steps involved can be found in Tab. 4.8.

Table 4.7: Training and render times on the GARDEN (images resolution: 2594×1681) and PLAYGROUND scene (1920×1080).

Method	Train	Render(GARDEN)	Render(PLAYGR.)
InstantNGP	0.25h	131ms	172ms
Mip-NeRF360	36h	38000ms	18000ms
ADOP	8h	30.3ms	14.5ms
Gaussian Spl.	0.75h	11.5ms	8.6ms
Ours	4h	16.4ms	11.1ms

Table 4.8: Breakdown of the frame time for the PLAYGROUND scene. Our method's "Rasterize" consists of: counting and memory allocation with 1.9ms, splatting with 2.6ms and combined sorting and blending with 1.7ms.

Method	#Points	Rasterize	Network	Tonemap	In Total
ADOP	12M	3.1ms	11.0ms	0.4ms	14.5ms
Gauss. Spl.	2M	8.6ms			8.6ms
Gauss. Spl.	8M	11.4ms			11.4ms
Ours	12M	6.2ms	4.5ms	0.4ms	11.1ms

4.3.5 Outlier Robustness

As seen in Fig. 4.9, our approach is robust to outlier measurements, for example, people walking through the scene. Especially volumetric approaches like MipNeRF-360 suffer from severe artifacts in this case, due to strong view-dependent over-fitting capability.

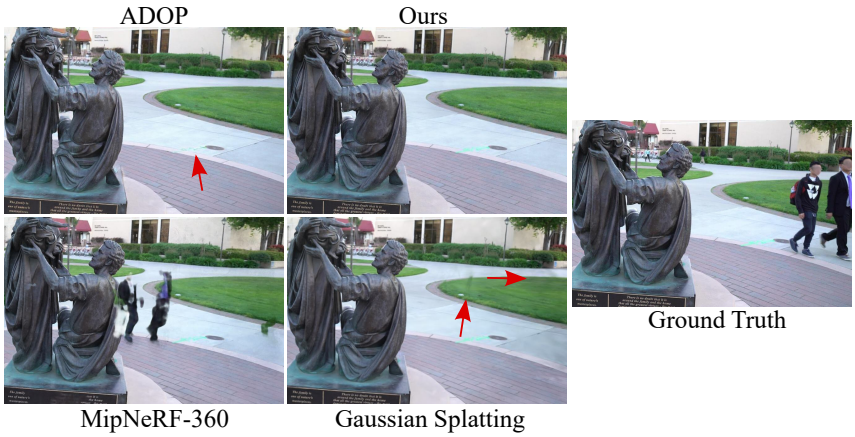


Figure 4.9: Comparison of outlier robustness on the FAMILY scene. Only our methods is able to remove floating artifacts while still retaining full color precision on the sidewalk.

4.3.6 Comparison to Prior Work with Number of Points

We have seen in previous experiments that 3DGS [Kerbl et al. 2023] has blurrier results compared to TRIPS, which can be confirmed by their weak LPIPS scores. However, they start with fewer point primitives (the SfM reconstruction) and thus are limited in the amount of detail to display. To this end, we conducted an experiment, where the 3DGS pipeline is provided with the dense point cloud (providing the same input as for our pipeline). 3DGS has a pruning mechanism to remove unwanted Gaussian, thus after their full training, from the initial 12.5M points only around 8M survived.

Table 4.9: Performance of the methods on the PLAYGROUND scene. Gaussian (dense) starts with COLMAP's dense reconstruction of 12M points and prunes them to 8M, Gaussian (sparse) is the original sparse setup and has about 2M points. Also see Fig. 4.10.

Method	LPIPS↓	PSNR↑	SSIM↑	Time ↓
Ours	0.229	25.12	0.746	11.1ms
ADOP	0.233	24.86	0.753	14.5ms
Gaussian *(dense)*	0.283	24.06	0.773	11.4ms
Gaussian *(sparse)*	0.322	24.61	0.776	8.6ms

Figure 4.10: Visual results of 3DGS with COLMAP's dense point cloud as input compared its normal setup as well as ours, which provides the sharpest results (PLAYGROUND scene).

The results of this experiment are presented in Tab. 4.9. It can be seen that LPIPS improves with more Gaussians (however, PSNR declines) as fine details can be reconstructed better. The qualitative comparison paints the same picture (see Fig. 4.10), where the quality of the grass improves drastically; however, finer details such as the chains still can only be reconstructed by us. Overall, the technique cannot reach the quality and scores of TRIPS, as we can keep more points to render efficiently as well as use neural descriptors to encode more detailed information.

Furthermore, our approach performs more efficiently in scenarios with large point clouds. In the dense setup, TRIPS outperforms 3DGS, as the resolution-dependent computation cost of our neural network (4.5ms at 1920×1080) catches up with our more efficient point rasterizer (see Tab. 4.8).

4.4 Limitations

In the preceding section, we demonstrate TRIPS' effectiveness on commonly encountered real-world datasets. However, we also identified potential limitations. One such limitation arises from the prerequisite to have an initial dense reconstruction (in contrast to 3DGS), which may not be practical in certain scenarios.

Additionally, our lack of an anisotropic splat formulation can create problems: When our method is tasked with strong hole filling of elongated, slender objects (such as poles), noisy artifacts surrounding their silhouettes can be observed. An example of this is depicted in Fig. 4.11. In such instances, the slightly blurred edge characteristic of 3DGS is often preferred.

Furthermore, even though the temporal consistency compared to previous point rendering approaches [Aliev et al. 2020; Rückert et al. 2022] has been drastically improved, slight flickering can still occur in areas with too many or too few points.

Our trilinear point splatting splits up points into distinct layers and as such loses depth information. Theoretically, during recombination, this could create holes in solid geometry. In practice, we could not find instances of this happening except in extreme zoom-ins far outside the training data. We believe that the per-point descriptors, the point inclusion in coarse layers, and the network-based recombination are capable of combating this issue, as reflected in the rendering quality.

Furthermore, as will be discussed in the next chapters, large-scale scenes and VR rendering are difficult with TRIPS.

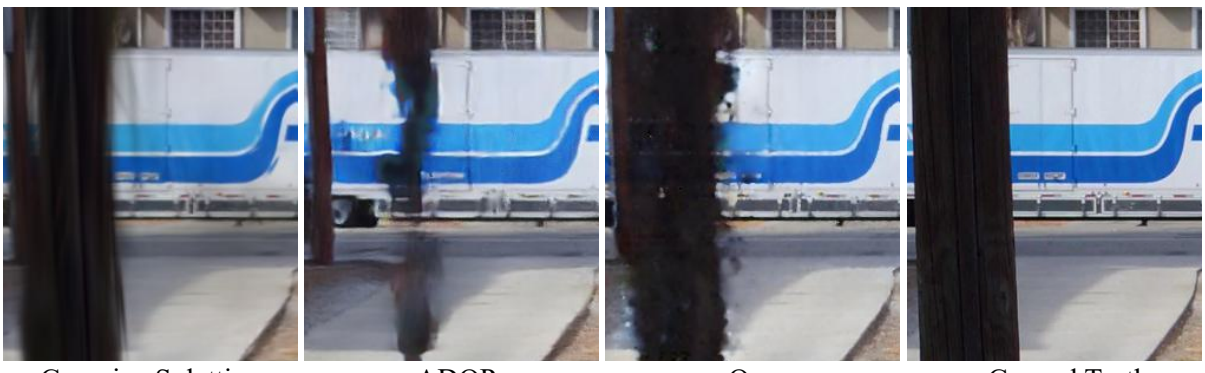


Figure 4.11: Limitation: Holefilling close to the camera exhibits fuzzy edges and shine-through.

4.5 Conclusion

In this chapter, we present TRIPS, a robust real-time point-based radiance field rendering pipeline. TRIPS employs an efficient strategy of rasterizing points into a screen-space image pyramid, allowing the efficient rendering of large points and is completely differentiable, thus allowing automatic optimization of point sizes and positions. This technique enables the rendering of highly detailed scenes and the filling of large gaps, all while maintaining a real-time frame rate on commonly available hardware.

TRIPS achieves high rendering quality, even in challenging scenarios such as scenes with intricate geometry, large-scale environments, and auto-exposed footage. Moreover, due to the smooth point rendering approach, a comparably simple neural reconstruction network is sufficient, resulting in real-time rendering performance.

CHAPTER 5

NePO: Neural Point Octrees for Large-scale Novel View Synthesis

This chapter is based on the paper Lewis et al. [2025], currently under submission. The author of this thesis contributed the algorithmic idea, project conceptualization, and supervised Noah Lewis during the implementation in their Master's thesis. In addition, the author of this thesis evaluated parts of the proposed method and competing methods and wrote the original draft of the paper. Noah Lewis contributed the implementation and major parts of the method evaluation and validation. Darius Rückert contributed implementation advice and hardware support during the evaluations. Marc Stamminger administered and helped conceptualize and supervise the project as well as provided critical feedback in all stages. All co-authors contributed review and editing during writing.

5.1 Introduction

TRIPS effectively renders scenes and consistently optimizes scene parameters. Nevertheless, hardware constraints limit point cloud sizes since having excessive points can cause GPU memory overflow. We aim to facilitate unrestricted exploration in vast scenes, since otherwise immersion can suffer if exploration is constrained by invisible barriers. Large-scale captures, however, using thousands of images aren't directly supported and often need partitioning or more intricate training methods, as mentioned by Turki et al. [2022]. Point-based methods like TRIPS encounter significant challenges, including high memory usage during training and inference, quality loss from vanishing gradients (notably in splatting methods), and substantial performance degradation due to overdraw or expensive splat sorting.

An example are datasets from LiDAR- and camera-equipped cars with billions of LiDAR points and thousands of ground truth images. NVS based on such data offers significant potential for numerous applications; however, they present challenges for existing methods. First, captured images exhibit a large depth range and spatial resolution variation, which requires specialized optimization and rendering techniques [Kerbl et al. 2024]. Second, these capture scenarios often involve limited viewing

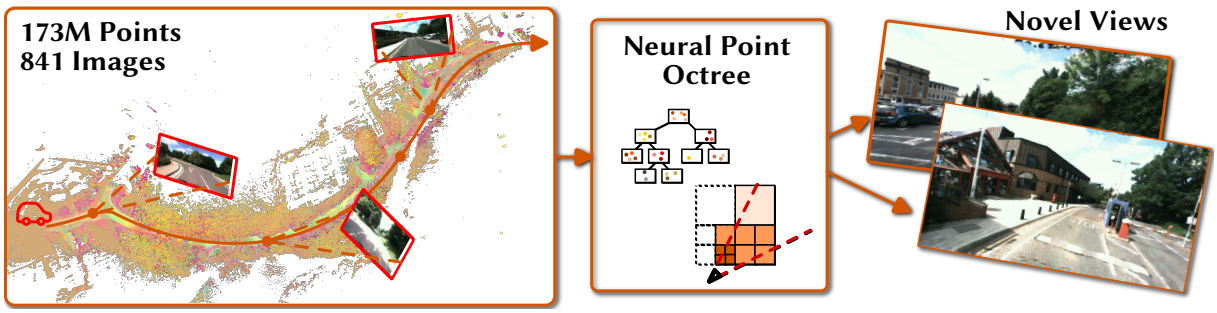


Figure 5.1: Input to our approach is a dense point cloud, as for instance achieved by LiDAR sensing on board a car, plus many, roughly posed RGB images (left). We compress the point cloud data into a neural point octree and optimize its features jointly with the camera parameters (center). After training, we can render novel views in high quality at more than 150 frames per second (right).

angles for parts of the scene, causing many prior methods to overfit to the captured views and fail for extrapolated view directions.

To overcome such limitations, we propose NePO, a hierarchical differentiable point-based radiance field rendering technique, based on Point Octrees (PO-Trees) [Schütz 2016]. The approach is tuned for large-scale LiDAR datasets, but the input point cloud can also come from other depth sensors or an MVS reconstruction. NePO optimizes radiance fields and enables high-quality real-time NVS, even for challenging datasets with hundreds of millions of LiDAR captured points without downsampling or data partitioning strategies [Kerbl et al. 2024; Lin et al. 2024a].

The underlying data structure of NePO is a neurally extended version of PO-Trees [Schütz 2016; Schütz et al. 2020], which provides intuitive LoD rendering and culling. We start with a dense point cloud and numerous posed ground-truth views with rough poses. Like PO-Tree, we generate a point octree in a bottom-up manner, sampling points for upper nodes from the full point cloud. Neural descriptors are assigned to the entire hierarchy such that upper nodes capture the global appearance and lower nodes add detail.

For rendering and training, the octree is cut and culled to ensure similarly sized projected bounding boxes on screen. An optional point budget controls memory usage based on hardware capabilities. Points are efficiently rendered as one-pixel sized splats and we output an approximate depth map, which is used to smooth the renderings temporally. Finally, we use a CNN to decode the descriptors and fill the remaining gaps in undersampled areas.

Our approach is fully differentiable and can optimize neural point descriptors, camera poses, as well as the back-end network and a final tone mapping stage. Furthermore, our point representation can be further densified and pruned based on gradient variance inside the octree nodes.

Our approach bears similarities to the hierarchical Gaussian Splatting approaches [Kerbl et al. 2024; Lin et al. 2024a], but has algorithmic differences. NePO optimizes a single hierarchical data structure for the entire scene, so no partitioning and consolidation step is necessary. Because our approach does not render overlapping splats, it exhibits less overdraw. In combination with a parallel LoD-aware multiresolution selection, we achieve very fast point rendering. However, the back-end reconstruction network has some performance impact, but it can reproduce delicate texture structures better.

In the results, we show the effectiveness of our approach on various data sets, allowing NVS with point clouds of hundreds of millions of points at real-time frame rates. We compare our approach with other state-of-the-art methods, showing superior quality and/or render times, in particular for scene data based on LiDAR.

The contributions of this chapter are as follows.

- A memory-efficient neural point-octree representation for real-time NVS and pose refinement of large-scale scenes.
- A novel point octree generation method optimized for neural rendering to incorporate large amounts of data from LiDARs.
- A novel, direct and differentiable rendering solution for this data structure that outputs a multi resolution feature pyramid.
- A detailed analysis of NePO's performance and rendering quality and comparison with prior work.

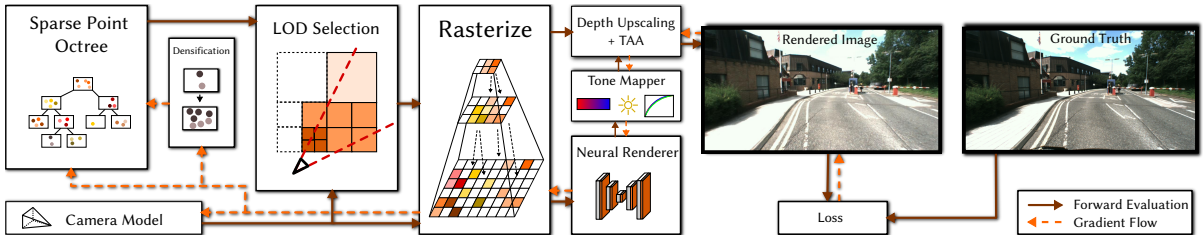


Figure 5.2: Our pipeline: We preprocess our unstructured point cloud to an octree. For rendering and optimization, we cut the octree based on LoD considerations and rasterize into multi-resolution feature images with progressively lower resolutions. A neural renderer and subsequent tone mapper resolve and combine the image pyramid. Following a temporal anti-aliasing module smooths the rendering. During training, all points as well as camera and tone mapping parameters are optimized, and the octree cells are further densified and pruned.

5.2 Related Works

For large scale reconstructions, recent methods subdivide the scene into blocks and optimize a NeRF for each of them [Duckworth et al. 2024; Mi and Xu 2023; Tancik et al. 2022; Turki et al. 2022], but do not provide LoD rendering and training. Mega-NeRF [Turki et al. 2022] and Switch-NeRF [Mi and Xu 2023] are based on an LoD-structure and provide great results, however rendering takes seconds per frame to compute.

In the context of explicit point-based rendering, approaches using data partitioning schemes [Lin et al. 2024a] or data distribution models [Chen and Lee 2024] show great results. Including both, Kerbl and Meuleman et al. [2024] introduced a highly efficient conquer-and-divide scheme called Hierarchical 3D Gaussian Splatting (Hier3DGS), optimizing smaller Gaussian representations independently and merging them into a hierarchical rendering structure (BVH).

5.3 Overview

An overview of the rendering pipeline is illustrated in Fig. 5.2. The input to the method are posed RGB images and a dense point cloud, which is first compressed into an octree structure (Sec. 5.4.1). Each point is augmented with four channel neural feature descriptors for appearance representation. Based on the camera parameters, the correct LoD is selected (Sec. 5.4.2) and rasterized in a multi-resolution fashion (Sec. 5.4.3). During training, the point cloud including neural descriptors, the neural network, and camera parameters are jointly optimized, based on our differentiable octree rendering (Sec. 5.4.4). Finally, a hierarchical neural network is applied to decode features and fill holes, and a tone mapper converts the result to the output image (Sec. 5.4.5). For scene rendering during inference, we adapt a temporal anti-aliasing scheme to eliminate flickering (Sec. 5.4.6). Furthermore, new points are spawned in incomplete regions (Sec. 5.4.7) and unused points are automatically deleted (Sec. 5.4.8). During training (Sec. 5.4.8), we optimize the neural renderer, point features, tone mapping parameters, and camera poses. All stages of the pipeline are implemented very efficiently with streaming in mind, thus arbitrary large scenes can be handled by our method (Sec. 5.4.10).

5.4 Neural Point Octrees

The foundation of our approach are point octrees (PO-Trees) [Schütz 2016; Schütz et al. 2020], a well-established technique to efficiently rasterize large amounts of points for *visualization*. The core idea is

that each hierarchy level contains a sub-sampled version of the original point cloud and each additional layer adds more points to fill holes and add more topologic information. Rendering is performed by top-down tree traversal and stopped when the required LoD level is reached.

5.4.1 Non-Uniformly Distributed Octree Generation

Our hierarchical data structure is based on PO-Tree. Also its construction follows the original ideas, but is slightly adapted to better handle the type of datasets we are aiming at.

First, we voxelize our point cloud and retain only one point within a very close area (typically 1 mm), effectively removing very small, dense clusters of points. These clusters do not enhance the quality of NVS in our system and would reduce performance due to overdraw. Next, we eliminate empty nodes from our representation and merge voxels, until each node contains a minimum number of points M. These sets of points $\mathcal{S}_0$ form the leaf nodes of our octree.

We then construct the tree in a bottom-up manner by progressively removing points from each node and inserting them one level higher. This step ensures that the point distribution in a parent node closely resembles the distribution of its eight child nodes, allowing upper nodes to serve as coarser approximations of the point distribution below. To achieve this, we first calculate the average distance d of all points in the point cloud. For each level l, we compute the corresponding sample distance d_l using the formula:

$$d_l = d \cdot 2^l \tag{5.1}$$

This distance is then used to determine the set of points $\mathcal{S}_l$ for each parent node from its eight child nodes $\mathcal{S}_{l-1}^i$ using Poisson disc sampling P with minimum distance d_l :

$$\mathcal{S}_l = \{P(\mathcal{S}_{l-1}^0, d_l), \ldots, P(\mathcal{S}_{l-1}^7, d_l)\} \tag{5.2}$$

Notably, Equ. 5.1 normally results in a quadratic (not cubic) sparsification per layer, as our point cloud originates from surfaces rather than volumes. This process creates an octree where each node contains a similar number of points or is empty.

5.4.2 Level of Detail Selection

To determine the nodes required of for a given view, we could perform a traversal of the hierarchy, determine for each node whether it is to be selected for rendering, and skip subtrees of not selected nodes. Because this sequential approach maps badly to GPUs, we instead iterate over all nodes of the hierarchy in parallel, evaluate the below described selection criterion independently, and gather all selected nodes in a list. This approach maps much better to highly parallel GPUs, even if it disables the skipping of entire subtrees. More details can be found in Sec. 5.4.10.

To decide whether a node is to be rendered, we project all corner points from n's bounding box to screen space and approximate its screen area $A(n)$ by the area of the screen space bounding box. Along with the node visibility function $\mathcal{V}(n)$, which returns false if n is off screen, our LoD selection criterion $C_{LOD}(n)$ for rendering a node is

$$C_{LOD}(n) = \mathcal{V}(n) \wedge A(n) > \delta, \tag{5.3}$$

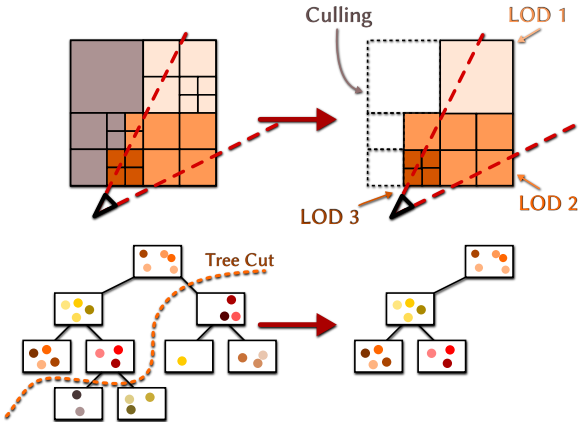


Figure 5.3: Our LoD scheme. We cull nodes outside the view frustum and if the projected node becomes too small. The resulting tree cut is then used for rendering.

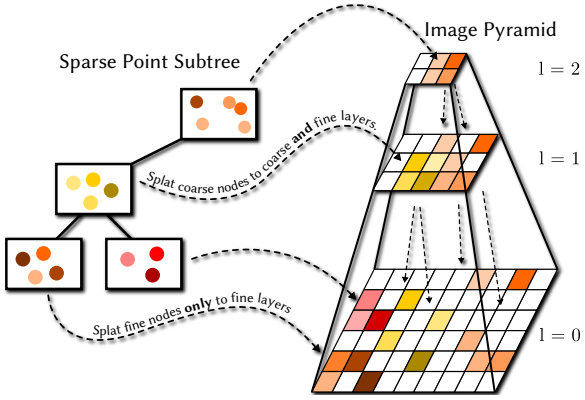


Figure 5.4: LoD-aware multi-resolution rendering. To avoid overdraw, we make our culling criterion progressively harsher, so that less nodes are drawn for lower resolution layers.

where δ is a threshold area in pixels. We set δ proportional to the desired node's minimum point number M to achieve a uniform point distribution across the screen. See Fig. 5.3 for a visualization.

Optionally, we can add a total point budget to the rendering, and cull lower level nodes once the budget is exceeded. This addition allows easy accommodation of different hardware and rendering demands.

5.4.3 Multi-resolution Rendering

As demonstrated in previous work [Aliev et al. 2020; Rückert et al. 2022], rendering the point cloud at multiple image resolutions significantly improves the final quality of neural rendering. With the Neural PO-Tree data structure, multi-resolution rendering can be seamlessly integrated into the rendering pipeline by adjusting the cutoff δ for each target layer. The multi-resolution LoD selection criterion for a node n at level l

$$C_{LOD}(n,l) = \mathcal{V}(n) \wedge \frac{A(n)}{4^l} > \frac{\delta}{u^l} \qquad \text{with } l \in [1\dots4] \tag{5.4}$$

ensures that low-level nodes (near the leaves) are rendered only to high-resolution targets, while high-level nodes are also rendered to the coarser levels of the image pyramid (see Fig. 5.4). Here, u is a tunable parameter, which we empirically set to 3 (see Sec. 5.5.4). This approach enhances and regularizes our representation, ensuring that coarser rendered layers as well as octree nodes capture the broader appearance, while finer layers and nodes add detail to the image.

5.4.4 Differentiable Rasterization

Our hierarchical representation is rasterized into an image pyramid with target images of varying resolution. After evaluating the LoD selection for each target level, all points of the selected nodes are rasterized as one-pixel sized splats. First, a fuzzy depth test [Rückert et al. 2022; Schütz et al. 2021] is performed as a pre-pass, allowing multiple points per pixel to pass within a small interval. Each point that passes this test is rasterized again, where descriptors of all points passing the fuzzy depth test are averaged. To obtain gradients with respect to the camera model and point positions, we use the approximate gradients method from Rückert et al. [2022]. Note that, because splatting is done at a pixel level, we support distorted and fisheye projection models out-of-the-box.

Due to the order of the selection test, we essentially rasterize the octree top-down, which allows efficient use of early-z testing, discarding points if occluded from rasterized coarser octree nodes. Furthermore, compared to unstructured point rendering [Rückert et al. 2022; Schütz et al. 2022], we are able to load and process data more coherently, which maps well to current GPUs.

The method centers on extensive LiDAR datasets where precise surface points are obtained, opting for ADOP-style depth test rendering over TRIPS' or VET's volumetric formulation (for details, refer to Chapters 3 and 4).

5.4.5 Neural Post Processing

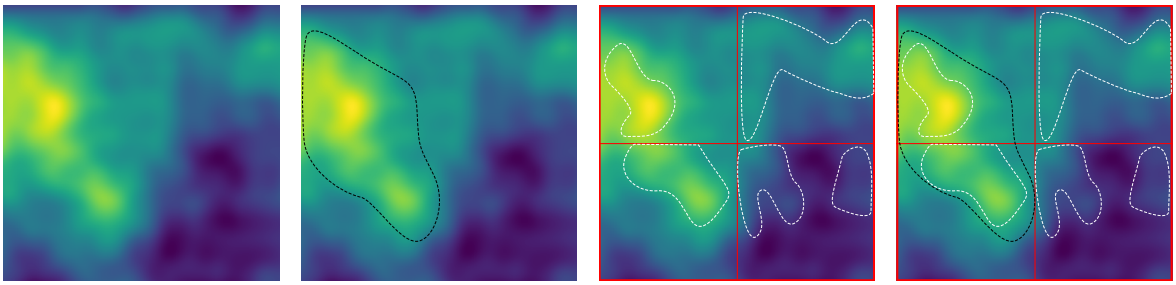
We follow the established approach of using a CNN for combined hole filling and feature decoding [Aliev et al. 2020], employing a network with four layers, skip connections, and average pooling. We utilize gated convolutions [Yu et al. 2019], which consist of a standard 2D convolution paired with a 2D masking convolution. This masking convolution is optimized to identify and disregard irrelevant information, and as a side benefit, it can be used for point pruning (see Sec. 5.4.7). The multi-resolution rendered feature maps are concatenated at their respective layers to match the network's resolution. After the neural network, a physically-based tone mapper [Rückert et al. 2022] is used to refine the output, addressing common issues in real-world captured datasets such as exposure and white-balance differences between images.

5.4.6 Temporal Anti-aliasing and Depth Buffer Reconstruction

Multi-resolution one-pixel point rendering in combination with convolutional neural network filtering is prone to flickering (see Chapter 4 and Rückert et al. [2022]). To solve this, we adapt temporal anti-aliasing (TAA) [Yang et al. 2020b] for our pipeline. Classical TAA works by reusing previous subpixel samples from previous frames by reprojecting them to the current frame, thus effectively supersampling the rendering. In our case, we want to supersample to avoid pixel samples flickering with small movement, either due to unoccluded points or due to nonlinearities in the network causing sudden color shifts. Thus our module's effect more closely resembles temporal smoothing.

To do so, we reproject pixel from the previous frame using the current and previous poses and rectify the color by clamping against the min/max color box of the 5×5 neighborhood of the destination. Colors are then accumulated with an exponential moving average: $C = \alpha C_{new} + (1 - \alpha) C_{old}$ with $\alpha = 0.1$.

Doing this filtering on the sparse point renderings, however, leaves pixels without valid depths. These pixels then still flicker, as it changing pixel footprint is not caught by TAA. Using a coarsely rendered depth map (e.g. our lowest rendered depth) does not work well, because its accuracy is too low for stably smoothed results. To solve this, we exploit our renderer's multi-resolution output and use hierarchical coverage-based depth upsampling [Grossman and Dally 1998] to reconstruct an approximate (but filled) depth buffer, which is then used for sample reprojection.



(a) Gradient magnitude (b) Top Gradients Parent (c) Top Gradients Children (d) Total Desification

Figure 5.5: Simplified 2D representation of (a) gradient magnitudes across a scene with marked areas of (b) where a parent node would duplicate points (black) and (c) where child nodes (red) would duplicate points (white). (d) represents all densified areas.

5.4.7 Densification

Real-world LiDAR point clouds often exhibit properties that impact NVS quality, commonly in the form of outliers or under-sampled regions, where LiDAR return signals were too noisy. In our renderings, this usually results in blurriness. Therefore, we densify and prune points in nodes during optimization. In contrast to Hahlbohm et al. [2025a] or Kerbl and Kopanas et al. [2023], this is not meant to reconstruct entire scenes from input images only, as we assume our LiDAR capturing to be of good quality. Similarly, using VET (see Chapter 3) is possible, however, it adds computational overhead, and would have to be scaled or chunked to avoid running out of memory on large-scale scenes.

Our densification strategy is therefore duplicating points with two purposes: First, this finely captures structures and edges. Second, this can increase appearance storage capacity for areas, as appearance is decoded from neural point descriptors.

We adaptively densify octree nodes whose descriptors exhibit strong gradients, as visualized in Fig. 5.5. Every 20 epochs, we average the absolute gradients of the descriptors g_i for all channels n over $t = 5$ epochs and compute a densification factor γ:

$$\gamma = \sum_{i=1}^{n} \left(\frac{1}{t} \sum^{t} |g_i| \right). \tag{5.5}$$

We then split the top 5% of points with the largest γ values, provided they also exceed $\gamma > \epsilon_d$, with epsilon set to 0.1 (ablated in Sec. 5.5.4).

5.4.8 Pruning

Unlike VET (refer to Chapter 3), we use a surface-based rendering approach without blending, which disables opacity optimization for pruning. Although it would be possible with NePO as well, rendering doesn't require it because our LiDAR points are surface-derived, eliminating the need for uncertainty volume rendering.

In our rendering, if a pixel is not hit by a single point, it is filled with the background color. Gated convolutions are trained to detect these holes and fill them using neighboring values. Outlier point descriptors are therefore automatically optimized towards the background value to prevent them from negatively impacting the rendering quality. Consequently, at the end of each epoch, we compute the

descriptor distance in feature space between each point and the background and remove any points where this distance is below a threshold $\epsilon_b = 1$. This method of outlier removal is efficient as it eliminates the need for explicit transparency optimization through sorted alpha blending [Kerbl et al. 2023].

5.4.9 Training

For training, we employ a combination of VGG_{19} and $\mathcal{L}_1$-loss and train each scene for 1000 epochs, where one epoch represents one iteration through all training images. Depending on the available GPU memory, we keep nodes in memory as long as possible to avoid additional transfer times. In the training stage, we also optimize tone mapping parameters to account for varying illumination. Using the approximately differentiable render approach of ADOP [Rückert et al. 2022], we also fine tune camera poses to improve consistency between the point cloud and the input images.

5.4.10 Implementation Details

To render large datasets with hundreds of millions of points in real-time, we implemented several improvements on existing techniques. Firstly, we evaluate the LoD for each node in parallel, which proved to be more efficient on GPUs compared to a top-down tree traversal. Additionally, we only launch threads for points in active nodes. To achieve this, the point cloud is sorted with respect to the octree structure, and after LoD selection, we compute a scan on the active nodes' point counts. Each thread can then calculate its node and point ID using binary search, which is faster and saves memory compared to explicitly storing it.

5.5 Evaluation

In this section, we evaluate our method on multiple datasets and compare it against related works, while also showcasing the effects of our design decisions.

5.5.1 Datasets

The datasets utilized in our evaluation are summarized in Tab. 5.1. MatrixCity [Li et al. 2023] is a vast synthetic dataset, portraying both aerial and driving scenarios, representing a complete cityscape with up to 400 million points. DurLAR [Li et al. 2021] is a real-world driving dataset, featuring LiDAR-equipped cars with poses obtained from the IMU. Office [Rückert et al. 2022] is a LiDAR and SLAM-derived dataset, highlighting challenges such as pose-drifts. The Boat dataset [Rückert et al. 2022] offers a diverse range of close-up and wide-angle shots, where RGB images are captured with auto-exposure, making it a valuable for testing the limits of concurrent work.

For scenes with no established train/test split, the commonly used leave-every-8th-out was used, for all others the dataset splits were used. Training times for ours are dependent on the amount of images used and range from 12h to 48h on a single RTX 4090.

Table 5.1: Scenes used in evaluation.

Scene	#Points	#Images	Resolution
Office [Rückert et al. 2022]	73M	688	1024×1024
Boat [Rückert et al. 2022]	53M	742	1440×960
DurLAR [Li et al. 2021]	173M	800	1024×544
MatrixCity [Li et al. 2023]	100M	8543	1920×1080
MatrixCity [Li et al. 2023]	403M	45101	1920×1080

Table 5.2: Quantitative evaluation. "X" indicates the method was not evaluated due to lack of resources.

	Office				DurLAR				Boat			
Method	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓	LPIPS ↓	PSNR ↑	SSIM ↑	Time↓	LPIPS ↓	PSNR ↑	SSIM ↑	Time ↓
3DGS	0.371	18.77	0.878	8.2ms	0.538	19.11	0.619	3.1ms	0.544	15.30	0.470	7.3ms
Hier3DGS $_{(\tau=0)}$	0.377	19.65	0.831	20.5ms	0.470	16.11	0.590	45.6ms	X	X	X	X
Hier3DGS $_{(\tau=9)}$	0.382	19.74	0.836	7.8ms	0.496	16.12	0.585	4.8ms	X	X	X	X
ADOP	0.279	21.47	**0.899**	22.4ms	X	X	X	X	0.301	20.49	0.650	18.7ms
Ours	**0.278**	**21.56**	0.877	8.6ms	**0.262**	**23.44**	**0.767**	5.5ms	**0.287**	**20.78**	**0.661**	10.9ms

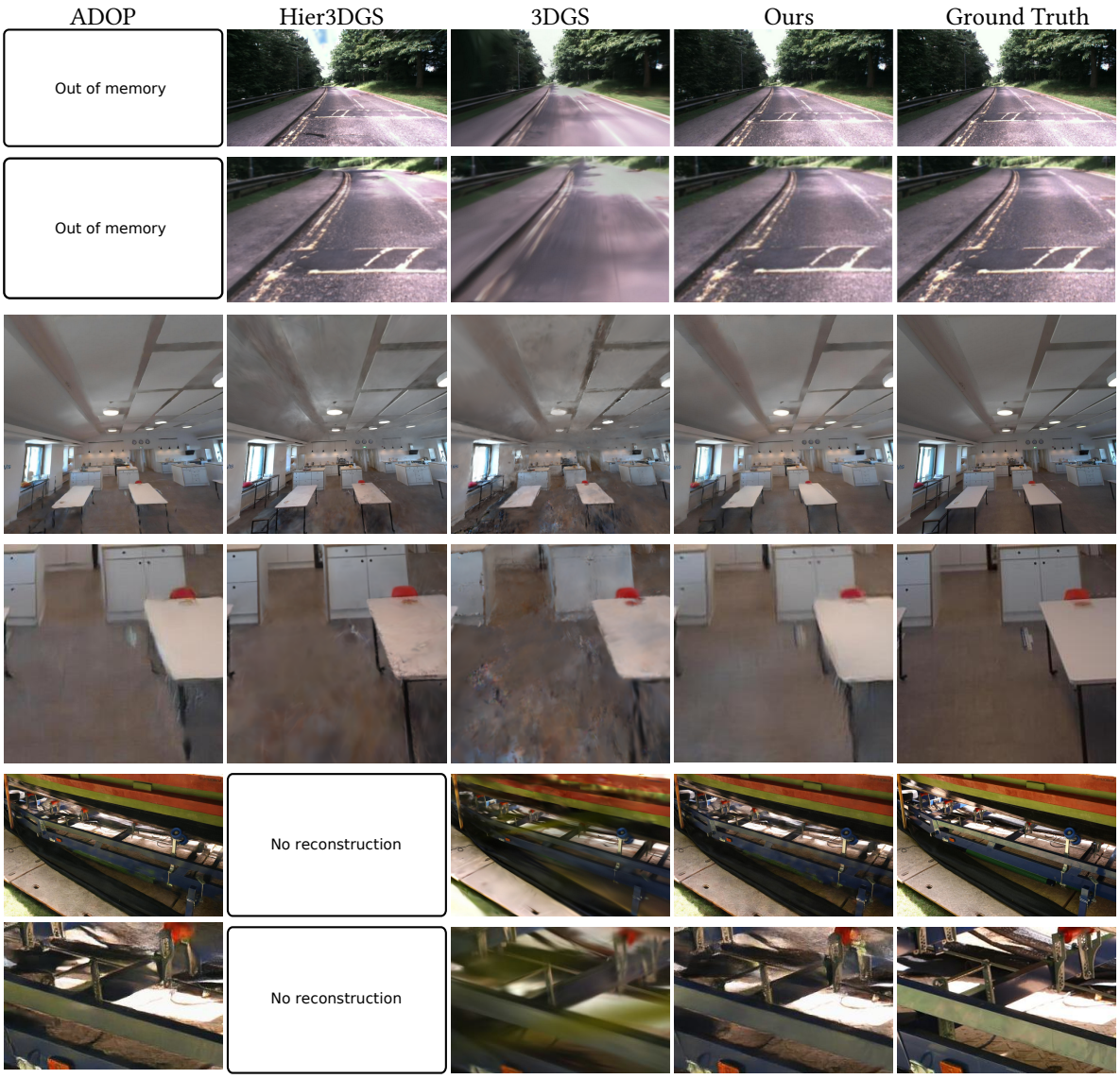


Figure 5.6: Visual comparison with ADOP, Hier3DGS, and 3DGS on DurLAR, Office, and Boat.

5.5.2 Comparison

We compare on all scenes, presenting the common evaluation metrics LPIPS, PSNR, and SSIM. We compare against the closest algorithm in terms of rendering paradigm (ADOP [Rückert et al. 2022]), as well as the recent 3DGS [Kerbl et al. 2023] and the hierarchical large-scale variant Hier3DGS [Kerbl et al. 2024]. All these approaches have open-sourced their code and aim for real-time rendering capabilities as us. Visual comparisons can be seen in Fig. 5.6.

Figure 5.7: Durlar scene: (top) Hier3DGS convergence issues, (bottom) different view with correct result.

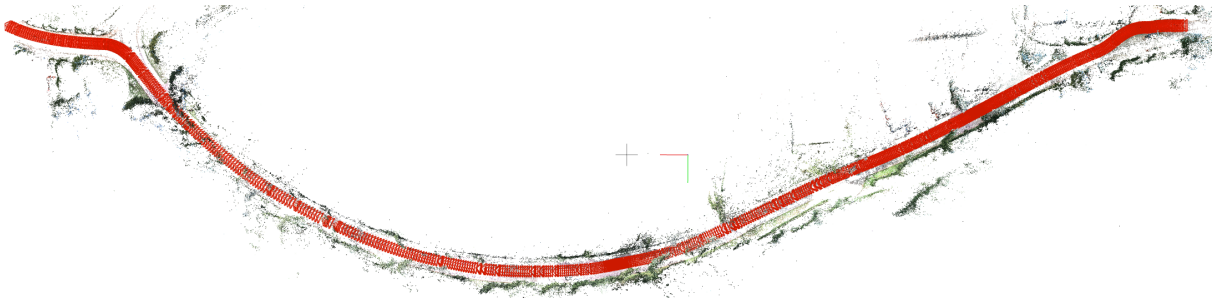


Figure 5.8: Durlar scene: COLMAP sparse reconstruction.

5.5.2.1 Real-world Scenes

Tab. 5.2 provides a quantitative evaluation of various methods on the Office, DurLAR, and Boat datasets. In these types of datasets, our method shines as it can use all available points. The results showcase the performance of each method in terms of visual quality and fidelity. Notably, our method achieves competitive scores.

On the multi-room Office dataset, our octree-based approach beats ADOP as baseline, as we can utilize the LoD scheme during training. Both Gaussian Splatting versions show difficulties with the sparseness of capturing location, which are further apart than on common datasets.

For car-based capturing, we evaluate the DurLAR dataset, which contains a long drive (see Fig. 5.1) with about 173M LiDAR points and over 800 images. ADOP fails due to memory limits, and 3DGS is outperformed by our method, in particular in the LPIPS metric, as seen in Tab. 5.2 (middle). For Hier3DGS, partially the method exhibits capturings with extremely poor quality, where parts of the Gaussians seem not to converge correctly (see Fig. 5.7), despite the availability of an accurate COLMAP reconstruction (see Fig. 5.8). To fairly compare, we take the 50 best images of Hier3DGS with none of these cases present and compare separately with our method in Tab. 5.3. Hereby, our method still shows better results.

On the Boat scene [Rückert et al. 2022], we slightly outperform ADOP and clearly outperform 3DGS. Comparing to Hier3DGS fairly was not possible on this scene, as their integrated COLMAP-based alignment was not able to register more than a quarter of all available images.

Table 5.3: Durlar scene: Best 50 images.

Method	LPIPS ↓	PSNR ↑	SSIM ↑
Hier3DGS	0.246	22.95	0.781
Ours	**0.223**	**25.47**	**0.820**

5.5.3 Runtime Performance Analysis

Tab. 5.4 presents a concise performance analysis of our method across different scenes, detailing the time taken for key stages in the rendering pipeline. The table highlights variations in processing times for octree cutting, depth processing, rendering point clouds, and U-Net refinement, with an overall time provided for each scene. Even on these large scenes with 400M points, our approach achieves real-time rendering performance with 18ms per frame. Point rendering itself is exceptionally fast and is dominated by the flat, per-pixel cost of our neural filtering backend.

Table 5.4: Performance analysis on an RTX 4090.

Scene	Octree Cut	Depth Prepass	Render Pass	U-Net	Overall
Office (73M)	0.6ms	0.9ms	2.1ms	5.0ms	8.6ms
DurLAR (173M)	1.2ms	0.6ms	1.0ms	2.7ms	5.5ms
Matrix City (100M)	0.6ms	2.6ms	3.9ms	11.1ms	18.2ms
Matrix City (403M)	1.3ms	2.8ms	3.4ms	11.1ms	18.6ms
Boat (53M)	0.5ms	1.0ms	2.4ms	6.6ms	10.5ms

5.5.4 Ablations

In the following, we present experiments to evaluate the modules of our method.

5.5.4.1 Multi-Resolution Rendering

Tab. 5.5 illustrates the results of employing different discard strategies. Higher values of u indicate more aggressive discard strategies, while lower values preserve more detail. From the results, we observe that setting $u = 4$ yields a good balance between quality metrics and computational efficiency and is also more robust on datasets with less dense point clouds. Stochastic discarding [Rückert et al. 2022] achieves similar rendering quality; however, rendering time is significantly increased.

Table 5.5: Lower level discard from Equ. 5.4.

	LPIPS ↓	PSNR ↑	SSIM ↑	Times ↓
No Discard	0.280	21.50	0.875	12.4ms
stoch. discard	0.278	21.52	0.878	11.0ms
u = 2	0.280	21.58	0.877	10.6ms
u = 3	0.280	21.55	0.877	10.1ms
u = 4	0.278	21.56	0.877	9.4ms
u = 8	0.280	21.64	0.877	9.2ms

5.5.4.2 Number of Points

Across point cloud sizes ranging from 1 million to 100 million points, our method exhibited varying quality. As seen in Tab. 5.6, on the synthetic matrix city dataset, we downsample the depth-fusion algorithm to create smaller point clouds. Our method has diminished quality when point densities are too low.

Table 5.6: Number of points and quality with the matrix city dataset.

# Points	LPIPS ↓	PSNR ↑	SSIM ↑	Times
1M	0.486	21.09	0.572	12.4ms
10M	0.352	24.55	0.710	13.4ms
20M	0.320	25.26	0.740	14.1ms
50M	0.290	25.96	0.772	16.4ms
100M	0.266	26.51	0.797	18.2ms

5.5.4.3 Level of Detail Cutoff

We use a LoD cutoff δ to cull nodes of a specific size on screen. As seen in Tab. 5.7, this cutoff reduces rendering times without impacting quality. This underlines the strength of our hierarchical structure.

Table 5.7: Ablation study: LoD cutoff δ from Equ. 5.3.

	LPIPS ↓	PSNR ↑	SSIM ↑	Times
1000	0.280	21.58	0.876	8.6ms
2000	0.281	21.57	0.877	8.3ms
4000	0.280	21.62	0.877	7.7ms
8000	0.281	21.63	0.877	7.2ms

5.5.4.4 Densification Factor

Our densification module is able to cheaply add points where descriptor resolution is too low. We ablate the different impact of the densification factor γ in Tab. 5.8. In general, we gain a small amount of quality on this LiDAR dataset with this method. We note however, that densification towards full scene reconstruction (as with Kerbl and Kopanas et al. [2023]) is not possible with this method.

Table 5.8: Ablation study: Densification factor γ from Equ. 5.5.

	LPIPS ↓	PSNR ↑	SSIM ↑	Added Points	Added Times
0.0001	0.276	21.40	0.876	+86.3M	+2.0ms
0.001	0.277	21.38	0.876	+86.2M	+2.0ms
0.01	0.276	21.42	0.877	+44.7M	+1.1ms
0.1	0.276	21.57	0.877	+0.3M	+0.0ms
no densify	0.278	21.56	0.877	-	-

5.5.4.5 Pruning Threshold

As we assume our input point cloud to be of good quality, our cheap pruning strategy (Seq. 5.4.8) discards points with feature vectors close to the background. As seen in Tab. 5.9, this is effective at removing small outliers and increases quality slightly. We choose $\epsilon = 1$ as our threshold.

Table 5.9: Ablation study: Pruning threshold ϵ_d.

	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	0.266	21.64	0.878
0.2	0.269	21.56	0.877
0.5	0.270	21.58	0.877
1.0	0.266	21.69	0.877

5.5.5 Large Scene Rendering

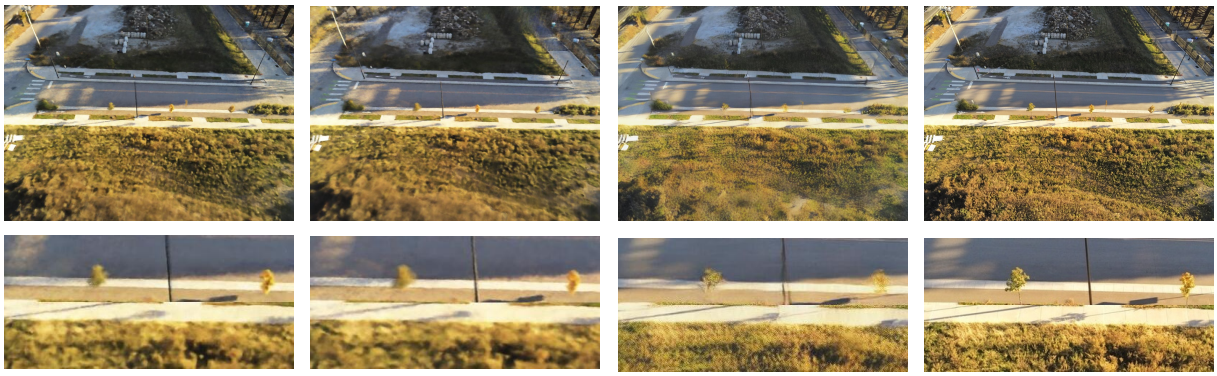
Rendering very large scenes is usually prohibitively expensive. For our approach, however, we showcase this capability with the full Matrix City dataset, containing over 45000 images and over 400M points. Our approach is able to render this gigantic scene efficiently at more than 50 fps. This setup of the scene was optimized for 50 epochs and took about 2 days.

5.6 Limitations

Although our method demonstrates effectiveness in handling large-scale datasets and producing high-quality results, it also exhibits limitations. Scalability remains a concern, particularly when dealing with extremely dense or complex scenes due to hard drive space constraints.

Additionally, the method's rendering speed may vary depending on resolution requirements, posing challenges for generalization. Furthermore, fast viewpoint changes, i.e., teleportation, lead to spikes in render times, as many nodes need to be loaded into memory.

Training times are also significant, especially for large datasets, requiring substantial computational resources. Furthermore, this method requires dense point clouds as input; if these cannot be provided (as can happen with MVS methods; see Fig. 5.9), tailored densification strategies such as VET (see Chapter 3) have to be included.



Switch-NeRF Mega-NeRF Ours Ground Truth

Figure 5.9: Visual comparison with Switch-NeRF and Mega-NeRF on the Rubble scene.

5.7 Conclusion

In summary, this chapter presents a robust and efficient method for neural rendering of large-scale point clouds. By leveraging neural point octrees we achieve high-quality results across diverse datasets. The approach addresses challenges in rendering massive datasets and offers real-time capabilities through optimizations such as parallel LoD evaluation and adaptive thread launching.

This chapter introduced NePO for depth-testing neural point rendering, which can also be expanded to volumetric rendering using TRIPS (Chapter 4) or VET (Chapter 3). However, for VR rendering, further rendering performance increases are necessary, as will be discussed in the next chapter.

CHAPTER 6

VR-Splatting: Foveated Radiance Field Rendering via 3D Gaussian Splatting and Neural Points

This chapter is based on the publication Franke et al. [2025]. The author of this thesis contributed the algorithmic idea and its implementation. Furthermore, he evaluated the proposed method and wrote the original draft of the paper. Laura Fink contributed to the visualizations, helped with user study methodology, and the supplemental video. Marc Stamminger supervised, administered, and helped conceptualize the project as well as provided critical feedback in all stages. All co-authors contributed review and editing during writing.

6.1 Introduction

For true immersive experiences, displaying real-world captured scenes on VR-headsets is the ultimate goal. However, state-of-the-art VR headsets exhibit high-resolution and high-refresh-rate displays, which place high and strict performance demands on rendering algorithms.

In light of this, in this chapter, we propose a method that allows interactive virtual scene exploration by exploiting the visual acuity falloff in the periphery through *foveated rendering*. Most foveated rendering algorithms render the foveal region as sharp and crisp as possible while trying to cut computational costs in the periphery by, e.g., lowering the resolution [Guenter et al. 2012] or the number of rays traced [Weier et al. 2016]. However, the application of such techniques is not trivial as they cause side effects such as temporal instability or flickering, which the peripheral vision is highly sensitive to [Weier et al. 2017].

As mentioned previously, 3DGS emerged as a volumetric scene representation and its volumetric nature inherently produces smooth renderings, and it lacks aliasing due to alpha blending of the primitives as well as low-pass filtering before rasterization. However, to achieve pleasing results with crisp details,

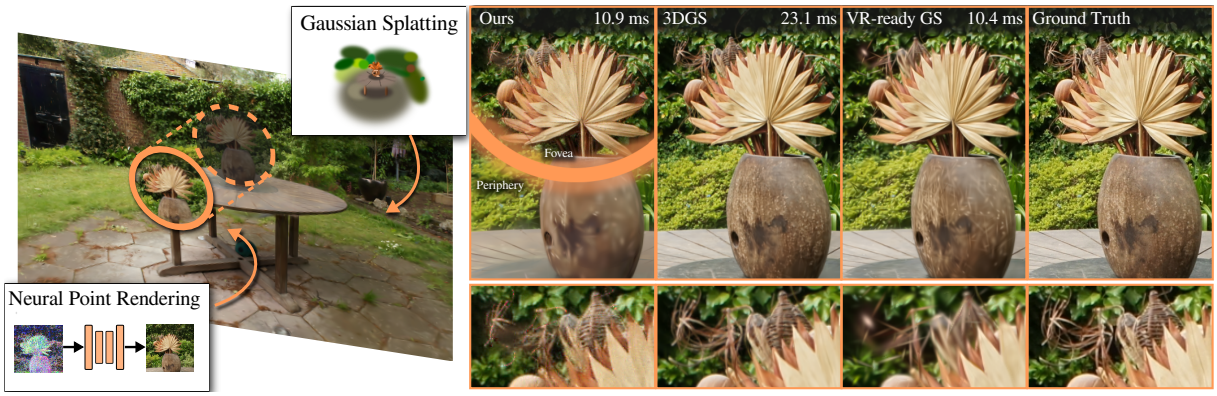


Figure 6.1: VR-Splatting combines advantages of neural point rendering (see Chapter 4) and 3DGS [Kerbl et al. 2023] within a hybrid foveated radiance field rendering system for VR. In contrast to 3DGS and a VR-ready variant, which only uses 0.5 Mio. primitives (see Fig. 6.2), our method provides high fidelity in the foveal region and meets the 90 Hz VR framerate requirements at 2016×2240 pixels per eye.

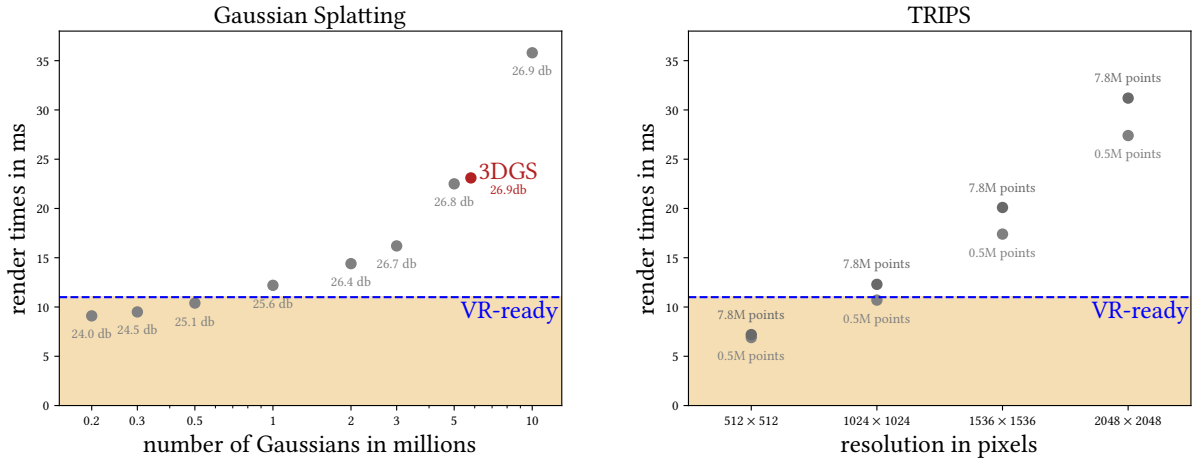
the required number of Gaussian primitives pushes the framerates below acceptable levels for VR, as shown in Figs. 6.1 and 6.2 (a). On the other hand, neural point rendering-based methods, like TRIPS (see Chapter 4), promise sharper results for the same time budgets. Both it and 3DGS use a Lagrangian scene representation, but model the splat sizes in different ways: 3DGS models splat size in Euclidian space via variance parameters, while the size of neural points is interpreted in pixel space using a neural network.

This causes the rendering performance to scale differently. 3DGS requires tiling and sorting of all primitives, and as such the rendering speed of 3DGS is reduced with an increasing number of primitives. In contrast, neural point rendering methods rasterize millions of primitives fast but are highly sensitive to resolution increases due to the CNN used for screen space filtering. Fig. 6.2 illustrates this behavior. Unfortunately, both methods do not meet the VR framerate requirements when used with full resolution and their default number of primitives.

In this juxtaposition, we identify a sweet spot matching foveated rendering: for the fovea, neural point rendering is able to cheaply render high-detailed, crisp small image crops. In particular, TRIPS has shown to be able to render fine details accurately (as seen in Fig. 6.3), which is advantageous for perceived quality given the magnification effects of current VR headset lenses. Regarding the periphery, 3DGS even with reduced number of primitives perfectly aligns with the demands of peripheral vision for high coherence and low flickering, as it renders a smooth but low detailed image that is temporally stable.

Instead of a purely 3DGS-based method, which reduces the number of primitives in the periphery, we opt for a hybrid approach that marries the individual advantages of the chosen methods regarding the reconstruction of the fovea and periphery. In this chapter, we thus introduce a hybrid system, which also allows us to further accelerate the system through synergies.

Given a VR headset with integrated eye tracking, our method first renders a low-detailed Gaussian rendering for both eyes for the full field of view. Here, we find that pixel-correct sorting [Radl et al.



(a) GS: number of Gaussians compared to render time in full VR resolution, with PSNR in db

(b) TRIPS: resolution compared to render time, with number of primitives

Figure 6.2: Rendering efficiency of 3DGS [Kerbl et al. 2023] and TRIPS (see Chapter 4). Left: Gaussian Splatting performance decreases drastically with increased number of primitives. To fit within VR limits, Gaussian counts need to be kept low, impacting quality. Right: In contrast, TRIPS performance scales mainly with resolution, allowing efficient rendering of high quality small crops.

Figure 6.3: Equal render time comparison for components of our method. Fovea-sized crops highlighted. For fine details, TRIPS often shows crisper results.

2024] is vital for smooth rendering. Using an occlusion estimation from this rendering together with querying the eye tracker, we cull the neural point cloud and rasterize it into an image pyramid for the foveal regions pixel footprint. This pyramid, together with the cropped Gaussian rendering, is then resolved with a small CNN, allowing the network to focus on adding fine detail in the foveal area only. The results of both renderings are then combined and sent to the VR headset.

This novel Gaussian and neural point representation is optimized from captured input images end-to-end, allowing virtual walk-throughs through real-world environments.

We contribute:

- A high-fidelity, 90 Hz capable technique using foveated radiance fields rendering for first-person VR rendering.
- The combination of neural point rendering and Gaussian rendering techniques for crisp foveal and smooth peripheral rendering.
- An evaluation of our foveated rendering method, including a user study showing its effectiveness.

An open source implementation is available under:

`https://github.com/lfranke/vr_splatting`

The project page including videos is available under:

`https://lfranke.github.io/vr_splatting`

6.2 Foveated and Virtual Reality Rendering

Exploiting the limitations of the human visual system (HVS) is a well established approach to relax rendering constraints and reduce computational demands [Wang et al. 2023a; Weier et al. 2017].

The Human Visual System. A key characteristic of human vision is the decrease in visual acuity in our peripheral vision, which foveated rendering techniques leverage. Weier et al. [2017] provide a comprehensive overview of the HVS and its exploitable limitations.

For designing rendering algorithms, the retina's physiology is crucial. The retina contains cones and rods that respond to light stimuli. The cones, mainly concentrated in the fovea (less than 10° of the visual field), are responsible for sharp, color vision [Goldstein and Brockmole 2016]. Rods, on the contrary, are motion sensitive and less color sensitive, dominating the peripheral retina with the highest concentration around 17° from the gaze direction [Curcio et al. 1990]. This distribution allows the human visual system to be modeled with a foveal region (*fovea*) for sharp vision and a peripheral region (*periphery*) sensitive primarily to brightness and motion. Exploiting this model to accelerate the rendering is called *foveated rendering*. Sometimes, it is assumed that the users gaze is always centered on the image, then called *fixed foveated rendering*. However, this assumption has failure cases and usually requires the foveal region to extend much larger than with eye-tracked methods. In this work, we employ eye tracking, as this allows us to keep the foveal region small and achieve better performance.

Foveated Synthetic Rendering. The concept of foveated rendering has been extensively investigated in the context of synthetic, artist-created scene rendering [Jabbireddy et al. 2022; Wang et al. 2023a]. Guenter et al. [2012] proposed generating three images per frame with full resolution in the fovea and progressively lower resolution in the periphery, blending them using bilinear upsampling. Nevertheless, this requires robust anti-aliasing to mitigate flickering, which can be reduced by decoupling visibility detection from shading [Patney et al. 2016].

Adaptive sampling in foveated ray tracing [Koskela et al. 2016] also reduces the computational load by focusing more samples on the fovea and fewer on the periphery [Friston et al. 2019; Stengel et al. 2016; Sun et al. 2017; Weier et al. 2016]. Stengel et al. [2016] use saliency maps to allocate samples effectively, while Weier et al. [2016] employ a reprojection scheme to accelerate ray tracing by resampling only visually important areas in the periphery, an idea also transferred to rasterization [Franke et al. 2021]. Other techniques include log-polar mappings for cost reduction [Meng et al. 2020b, 2018], contrast-aware foveation for ray tracing and rasterization [Tursun et al. 2019] and deep foveated reconstruction trained with video [Kaplanyan et al. 2019] or neural super resolution techniques [Ye et al. 2024b].

Further supporting research has explored methods to reduce perceptual artifacts. For example, adding depth of field [Weier et al. 2018], developing perceptual image metrics [Mantiuk et al. 2022, 2021; Swafford et al. 2016], exploring saliency [Sitzmann et al. 2018], exploiting eye domination [Meng et al. 2020a], hardware solutions [Kim et al. 2019] and examining peripheral acuity [Hoffman et al. 2018] have all proven promising. Eye-tracking latency requirements [Albert et al. 2017] suggest that a maximum latency of 50-70 ms is acceptable to avoid noticing artifacts, particularly during rapid eye movements (saccades).

Radiance Field Rendering. Radiance field rendering methods for VR are sparse, especially due to rendering budget constraints exceeding common methods.

In the domain of NeRFs, FoV-NeRF [Deng et al. 2022] introduces an egocentric NeRF variant with adapted sampling schemes for foveated rendering using multiple MLPs. Shi et al. [2024] extend this with a scene-aware formulation, whereby they improve the expressiveness for detailed areas. Wang et al. [2024] further augment this field with a voxel-based representation including a visual sensitivity model. These three approaches inherit the challenging inference performance characteristics of NeRFs,

as framerates stays below 90 Hz. Similarly, hash grid NeRF-based super-resolution techniques [Li et al. 2022] and variable rate shading approaches [Rolff et al. 2023a,b] achieve impressive results, but still fall short of the high framerate and resolution requirements of VR rendering. Consequently, Xu et al. [2023] introduce a multi-gpu rendering scheme, allowing 36 fps when using three A40 GPUs. Additionally, they rely on a space-carving scheme to lower ray-marching costs. Following, Turki et al. [2024] extend this with a hybrid representation reducing rendering costs for surfaces and only using full ray-marching for challenging regions.

Point-based methods also struggle with VR capabilities. Neural point-based methods such as ADOP [Rückert et al. 2022] enable VR rendering, however, only for low-resolution output. Likewise, the recently published official SIBR [Bonopera et al. 2020]-based VR implementation, as well as Gaussian Splatting extensions for Unity and Unreal, fail to render high-resolution details due to the amount of Gaussians required. Although some methods focus on VR applications [Jiang et al. 2024], they do not prioritize rendering efficiency.

6.3 Method

As input, we use a set of real-world images captured using, e.g., a smartphone or DSLR camera. For our method, we use SfM/MVS to obtain camera intrinsics, poses, and point clouds. We first present and evaluate the method core (Sec. 6.3.1), motivating the refinements with more sophisticated peripheral rendering (Sec. 6.3.2), accelerated foveal rendering (Sec. 6.3.3), and a fovea and edge-based combination function (Sec. 6.3.4). Our pipeline is optimized end-to-end from the input images, camera parameters, and point clouds (Sec. 6.3.5).

6.3.1 Pilot Study

The core idea is a deftly blend of Gaussian splats and neural points for foveated rendering. Similar to Patney et al. [2016] using a *perceptual sandbox*, we first implemented a simplistic form of this combination and evaluated it with a pilot study, in order to evaluate the potential of the idea and to identify problems that require further refinement. First, we present and discuss the findings of this first naive baseline. Subsequently, we continue with the description of our final system, including a number of improvements, which we developed based on the study findings.

Naive Method. To render a view, we render the point cloud in a foveated fashion, using fixation point information obtained from the VR headset's eye tracker.

For the periphery, we employ 3DGS [Kerbl et al. 2023]. Each splat has an opacity α and a color C. Their 3D orientation and scaling are parameterized per point via a covariance matrix Σ. These primitives are projected and blended to screen, resulting in a full-resolution image P. For the study, the splats were initialized from SfM and densified carefully during training to ensure low rendering cost, resulting in 0.4 million primitives.

For the foveal region, we employ TRIPS (see Chapter 4) initialized with a dense point cloud estimated by MVS. Each point has contribution size s and a compressed neural feature τ. The features essentially represent colors; however, their higher dimensionality allows encoding additional information. The points are projected and blended into a multi-resolution image pyramid, based on the projected size s (see Chapter 4) and then combined by a small CNN, resulting in a fovea-sized image F. For the pilot study, P and F were combined using a smooth blending function and the fovea size was set to 17°.

The parameters of Gaussians, neural points and CNN are optimized end-to-end with gradient descent from the input images with their respective proposed optimization functions.

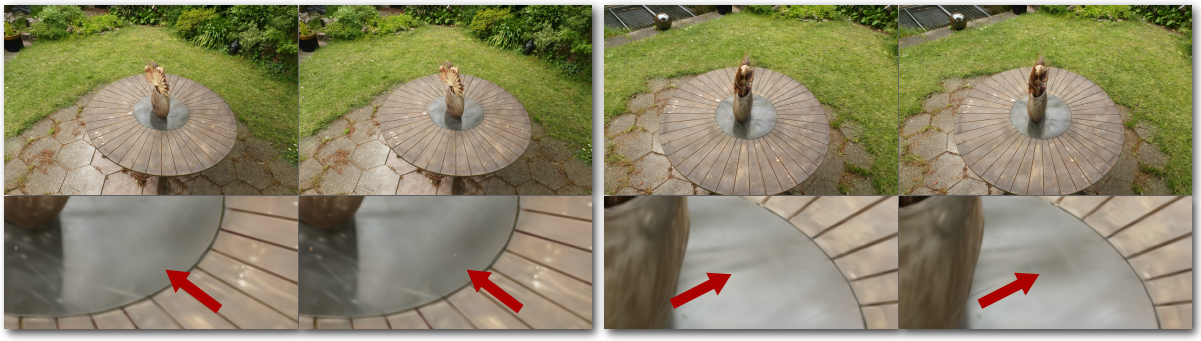


Figure 6.4: Gaussians in 3DGS "popping" in and out during small camera rotations, resulting in black spots suddenly appearing and disappearing. Popping is especially noticeable in VR, as revealed in our pilot study.

Pilot Study. We asked four computer graphics experts (aged 23-28, 1 female, 3 male) with VR experience for their observations and rankings on a VR-ready 3DGS version with 0.5M primitives (denoted as *VRGS*) and contrasted it with our naive baseline on the popular GARDEN scene from the MipNeRF-360 dataset [Barron et al. 2022].

The participants were asked to state their most important visual observations. Consistent for both stimuli, participants noticed distracting "popping" artifacts, caused by inaccurate depth sorting in 3DGS, as noted by Radl and Steiner et al. [Radl et al. 2024]. An example is shown in Fig. 6.4. The effect is amplified by the magnification of the VR headset's lenses and the reduced number of primitives.

Regarding VRGS, participants noted blurred splotches due to underreconstruction with the reduced number of Gaussians [Ye et al. 2024a]. More blurriness was also observed compared to the foveated method, which coincides with observations from previous works [Hahlbohm et al. 2025a].

For our foveated method, the quality of the visual experience was rated slightly lower compared to VRGS. Although some mentioned increased detail richness, the experience was overshadowed by artifacts. Several things were mentioned: mismatching color palettes as the VGG-loss used by TRIPS (see Chapter 4) is known to not force color fidelity [Zuo and Deng 2023]. Furthermore, the suboptimal response of eye tracking due to delays and blinking was criticized. Lastly, noise and temporal jittering was mentioned, also noted in related work [Aliev et al. 2020; Rückert et al. 2022].

The study confirmed that the performance cuts of the VR-ready version of Gaussian Splatting lead to apparent artifacts due to underreconstruction. Although the experience of the naive foveated implementation was lacking for users, it was rated at a similar quality level. However, the experts' critique showed that combining peripheral and foveal regions from two different representations is non-trivial.

Based on these findings, we identify three improvements: smoother Gaussian reconstruction for peripheral regions without popping artifacts (Sec. 6.3.2), fast foveal reconstruction with well-integrated eye tracking (Sec. 6.3.3), and a coherent mixture of the models including a matching color palette and subtle blending of high-frequency detail (Secs. 6.3.4 and 6.3.5). Our resulting pipeline can be seen in Fig. 6.5.

6.3.2 Peripheral Rendering

We identify three key properties which are required for our Gaussian-based peripheral reconstruction: (1) high rendering performance, (2) little-to-no popping artifacts, and (3) little underreconstruction.

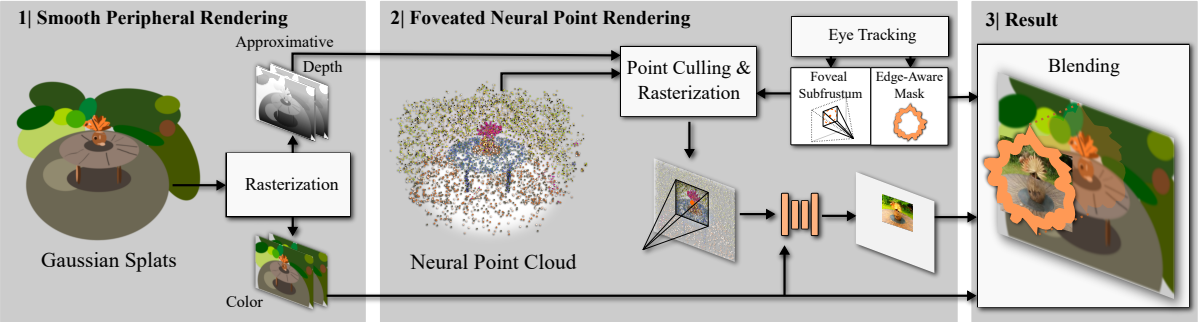


Figure 6.5: Our Pipeline. A smooth color image and approximate depth map are rendered from a limited set of 3D Gaussians with a temporally stable sorting active (see Section 6.3.2). Afterwards the eye tracking system is queried to construct a subfrustum via an adapted projection matrix covering only the foveal region. We project a separate neural point cloud with the adapted matrix. Points occluded by the denser Gaussian splats, are culled against the approximate depth maps and the result is processed by a small CNN (see Section 6.3.3). Eventually, we blend the peripheral color output and the foveal region using an egde-aware mask (see Section 6.3.4).

Usually, these three are conflicting goals meaning an acceptable solution for all three needs to be designed carefully.

Regarding (2), we adapt the sorting scheme of Radl and Steiner et al. [2024], which eliminates popping by hierarchically sorting Gaussian contributions per pixel (instead of per primitive) at the cost of increased processing time per Gaussian. Furthermore, by employing a more finely detailed Gaussian densification heuristic [Kerbl et al. 2024; Ye et al. 2024a], we reduce the severance of (3).

This Gaussian renderer is temporally stable also with regards to popping and without underreconstruction artifacts which are noticeable in the periphery. An example rendering can be seen in Fig. 6.6 (a). Regarding performance, we need to drastically reduce the number of Gaussians (as seen in Tab. 6.2), which we achieve by limiting the densification during training.

Note that we do not mask out the foveal region in this step and instead generate a full image. Furthermore, we generate an approximate depth buffer [Radl et al. 2024] by accumulating per-pixel depths d with

$$d = \sum_{i=1}^{n} d_i \alpha_i \prod_{m=1}^{i-1} (1 - \alpha_m). \tag{6.1}$$

While this depth buffer is not needed for the peripheral rendering, it is used to cull occluded points in the following foveal point rendering step.

6.3.3 Foveal Rendering

In our context of foveated rendering, we improve TRIPS (see Chapter 4) significantly by using information gained with the previous (full screen) Gaussian rendering results.

Firstly, we only render a subfrustum of the entire scene and discard points outside this frustum as early as possible. This subfrustum is constructed around the fixation point, which is queried from the eye tracker after the periphery has been rendered. Additionally, we use the approximate depth buffer rendered (Eq. 6.1) from the Gaussians to discard occluded points. This occlusion culling reduces rendering times, accelerates convergence, and eliminates point render flickering, as this simplifies visibility and blending computations.

Furthermore, we inject the fovea crop of the less detailed Gaussian rendering into the CNN. Thus, the network is only tasked with augmenting fine details, instead of balancing hole filling and crisp reconstruction as in TRIPS. As a side product, this severely reduces training cost.

These optimizations allow us to halve the number of filters for the full-resolution CNN layer to further accelerate our pipeline. Full-resolution convolutions are originally the most expensive part of the TRIPS network evaluation, taking up more than 70% of the network inference time.

6.3.4 Combination

Our renderer outputs two images P (peripheral) and F (foveal). It is important to carefully combine both images, as the visual acuity falloff of the human eye is gradual. Therefore, we introduce a combination factor c that includes the two terms f_p (positional) and f_e (edge-based) for each pixel (u,v):

$$r_{norm} = \text{clamp}\left(\frac{1}{d_f}\sqrt{(u-e_x)^2 + (v-e_y)^2}, 0, 1\right) \tag{6.2}$$

$$f_p = \frac{r_{norm} - m}{1 - m}, \tag{6.3}$$

with $e_{x,y}$ being the pixel position of the fixated point, d_f the fovea radius in pixels and m the start of the normalized blending interval between 0 and 1. We empirically use $m = 0.75$. We use d_f to 256 pixels (see Sec. 6.4), which is larger than the fovea's 7% eccentricity, however necessary due to inaccuracies of the eye-tracker.

The second factor is the edge factor f_e, based on the Sobel edge filter S of the point image F:

$$f_e = S(F, u, v). \tag{6.4}$$

f_e is inspired by Franke et al. [2021], who force color discontinuities in the transition between the fovea and the periphery to be gradual to avoid fast color changes.

Finally, the interpolation factor per pixel c is computed as

$$c = 1 - S_2(f_p + \gamma * f_e), \tag{6.5}$$

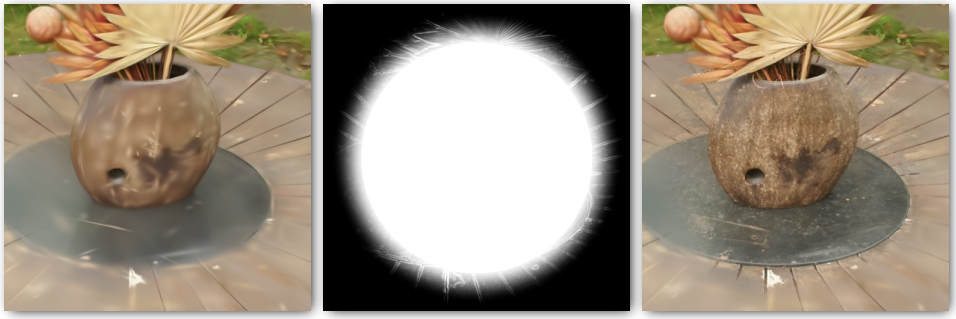
using $\gamma = 0.2$ and with S_2 the smootherstep function $S_2(x) = 6x^5 - 15x^4 + 10x^3$ for a smooth transition and well defined derivations suited for end-to-end training and smooth blending. The combined image is then computed for each pixel with $(1-c)p + cf$ with p and f the individual pixels of P and F.

To account for variance in image exposures, we then follow ADOP [Rückert et al. 2022] and use a differentiable camera model to tone-map the resulting images to a displayable LDR space.

6.3.5 Losses and Optimization

We optimize our pipeline end-to-end. For every iteration, we randomly sample a fixation point and evaluate the pipeline to arrive at a foveated image. We compare this image to the ground truth and optimize via gradient descent using our loss function $\mathcal{L}$:

$$\mathcal{L} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{D-SSIM} + \mu\mathcal{L}_{VGG}(F) + \beta\mathcal{R} \tag{6.6}$$



(a) Gaussian rendering P (b) Combination blending map (c) Our result combining P and F

Figure 6.6: Our combination mask (b) in effect (zoomed in). It is used to adaptively merge Gaussian output with neural point rendering.

The first two terms are identical to 3DGS [Kerbl et al. 2023] with $\lambda = 0.2$. We further augment this with the VGG19-loss [Ledig et al. 2017] for neural point rendering with $\mu = 0.001$. VGG-loss has proven to work well for neural point rendering; however, for Gaussian Splatting, it increases convergence time and causes artifacts with densification. Therefore we only apply it to neural point renderings. Lastly, we introduce a regularizer $\mathcal{R}$:

$$\mathcal{R} = |\overline{F} - \overline{\mathcal{F}(P)}|, \tag{6.7}$$

with the foveal cropping function $\mathcal{F}$. This ensures that the mean of F and the part of P with pixels inside the fovea are similar. This regularizer is vital to reduce the color shift, which optimizing with VGG loss otherwise induces. We empirically found $\beta = 0.00001$ to work well. Although a slight shift is still present, it isn't perceived by users.

We apply an opacity penalty to the opacities of both Gaussians [Radl et al. 2024] and neural points (see Section 3.4.3), which causes optimization to discard obsolete primitives.

6.4 Implementation and Optimization Details

We implemented our method in C++ and CUDA, using torch as the automatic gradient framework and followed the framework of Patas [2023] for a Gaussian Splatting implementation and Radl and Steiner et al.'s sorting algorithm [Radl et al. 2024].

Training. For training, we optimize Gaussians for 30000 iterations (as in 3DGS [Kerbl et al. 2023]) and neural points for 30000 additional iterations with fixed Gaussian parameters. For the first 2000 iterations, both systems are trained independently to avoid the faster converging Gaussians to interfere with neural point optimization. Afterwards, gradients from the foveal rendering are also propagated back to Gaussian parameters except for VGG-loss computations. Sampling a random eye position every iteration maps well to cropped training usually employed with CNNs. After 45000 iterations, we start regularizing the training with $\mathcal{R}$.

For densification, we lower the thresholds compared to 3DGS and use 0.0002 as densification gradient threshold, 200 as densification interval, 0.005 as opacity pruning threshold, and 0.999 as opacity decay every 50 epochs [Radl et al. 2024]. For the scenes we tested, this results in 0.3M to 0.5M Gaussians.

For neural points, we impose an opacity gradient penalty of 10^{-8} every update step (see Section 3.4.3), usually resulting in about 5M points.

We uniformly sample fixation points over the image, which is similar to TRIPS using a cropped training regime. The increased iterations hereby also ensure all parts of the scene to receive coverage during training.

Inference. For inference, we adjusted several aspects to VR rendering. We target the HTC Vive Pro Eye, having an eye-tracker capable of updating rates of 120 Hz. We bake all rendering attributes directly into the point cloud, such as opacities, features, and covariances to minimize runtime costs. Furthermore, we delay eye tracking updates until after rendering P, drastically reducing lag. Also, compute and evaluate all renderings in half-precision and evaluate the network with images for both eyes batched together using cuDNN for acceleration.

For neural point rendering, rendering only the foveal regions allows for algorithmic optimizations with respect to performance: TRIPS originally renders up to eight progressively lower layers individually, first by counting the pixel contributors, then scanning and allocating a buffer for splatting. These steps introduces overhead, which we reduce by combining all layer computations, speeding up the computations by about 2 ms. This requires about four times the GPU memory of the original implementation; however, this is negligible as rendering is limited to the foveal region.

Fovea Size in Pixels. The effective fovea size in pixels required for our formulation necessitates the estimation of pixels per degree of the field of view. This is complicated by the use of distortion lenses [Kreylos 2018] and is rarely accurately reported by headset manufacturers. We follow Kreylos' upper limit estimate of 15.7 pixels per degree [Kreylos 2019] for the headsets display system, which worked well in our tests.

Related works use fovea sizes between 7.5° and 17° [Franke et al. 2021; Patney et al. 2016; Weier et al. 2016]. Using the latter as the upper limit and the driver-initiated resolution scale of 1.4×, we arrive at a fovea size of around 400 pixels, which we round to 512 pixels (thus $d_f = 256$) for implementation reasons and to combat eye-tracking inaccuracies.

6.5 Evaluation

We evaluate our method on real-world captured scenes; we use the popular MipNeRF-360 dataset [Barron et al. 2022] as well as the INTERMEDIATE set from Tanks&Temples [Knapitsch et al. 2017].

6.5.1 Rendering Performance

We compare our methods rendering performance with the established NeRF-based approaches FoV-Nerf [Deng et al. 2022], Scene-aware Foveated Neural Radiance Fields [Shi et al. 2024] (SA Fov-Nerf), VR-NeRF [Xu et al. 2023] and HybridNeRF [Turki et al. 2024]. Furthermore, we include the differentiable point rendering techniques TRIPS (see Chapter 4) and 3DGS [Kerbl et al. 2023] on which our method is built.

In Tab. 6.1 we present the results of this overview, measured on a single RTX 4090, with metrics reported by related works [Shi et al. 2024; Turki et al. 2024]. Our performance calculations were performed by

rendering in full VR resolution, with the drive- recommended 2016×2240 pixels per eye. Our method is the only that surpasses the VR framerate limit of 11.1ms per frame, with speedups of more than 1.45× compared to the fastest related approach "SA Fov-Nerf", who report their timing only for resolutions of 2.44 megapixel compared to our 4.52 megapixel.

Table 6.1: Comparison of reported rendering times of related VR-capable radiance field methods on a single RTX 4090. Timing † from Shi et al. [2024] and ‡ from Turki et al. [2024].

	Fov-NeRF [Deng et al. 2022]	SA Fov-NeRF [Shi et al. 2024]	VR-NeRF [Xu et al. 2023]	HybridNeRF [Turki et al. 2024]	TRIPS (Chapter 4)	3DGS [Kerbl et al. 2023]	Ours
Resolution	1440 × 1700	1440 × 1700	2064 × 2096	2064 × 2096	2016 × 2240	2016 × 2240	2016 × 2240
Render times	23.5 ms †	16.1 ms †	165.3 ms ‡	21.8 ms ‡	33.1 ms	23.1 ms	10.9 ms

For further evaluation, we also compare to a reduced Gaussian Splatting variant with VR-ready performance using 0.5M primitives (see Fig. 6.2). We call this variant *VRGS*, which renders slightly faster than our method (as seen in Tab. 6.2).

For training speed, we improve drastically on TRIPS (see Tab. 6.2, right), however, both Gaussian Splatting variants converge faster as no neural network optimization is required.

6.5.2 Quantitative Evaluation

To quantitatively evaluate our method, we compare our method with the mentioned VRGS, our interpretation of a VR-ready version of 3DGS. Furthermore, we compare against 3DGS [Kerbl et al. 2023] and TRIPS (see Chapter 4) as qualitative baselines and closest to our method even though they are not fast enough for VR.

We compare using the common quality metrics $\text{LPIPS}_{\text{VGG}}$ [Zhang et al. 2018], PSNR and SSIM. For these metrics, we only evaluate the foveal region and omit the periphery.

The results are seen in Tab. 6.2, averaged per dataset. Our experiments show that our method is preferable in the LPIPS metric, which is more perceptually close to the human visual system [Zhang et al. 2018; Zuo and Deng 2023], while being close in the other metrics. Importantly, our method is not subject to popping.

For computing JOD (Just-Objectionable-Difference), we use FovVideoVDP [Mantiuk et al. 2021], which is a perceptional metric that includes terms for contrast sensitivity and peripheral vision. This method is designed and capable of metrically evaluating foveated rendering and scoring it in JOD between 0 and 10, the higher the better. We use it with the preset for the HTC Vive Pro (having the same display system we use) in the setup: *"FovVideoVDP v1.2.0, 13.15 [pix/deg], Lpeak=133.3, Lblack=0.1 [cd/m^2], foveated"*. The metric is computed including peripheral regions. As also seen in Tab. 6.2, our method consistently outperforms VRGS in this metric.

On the project page, renderings of our method can be seen during use with a VR headset.

Table 6.2: Quantitative metrics. JOD (Just-Objectionable-Difference) [Mantiuk et al. 2021] is computed with *FovVideoVDP* for an HTC Vive Pro in foveated mode for the full images. LPIPS, PSNR, and SSIM metrics are evaluated for the foveal regions only.

				MipNeRF-360				*Tanks&Temples*				
Target System	Method	No Popping	Performance	JOD↑	LPIPS↓	PSNR↑	SSIM↑	JOD↑	LPIPS↓	PSNR↑	SSIM↑	Training
Desktop	3DGS	×	23.1 ms	7.90	0.247	**27.01**	**0.804**	**8.24**	0.198	**27.15**	**0.874**	0.5h
	TRIPS	✓	33.1 ms	**8.06**	**0.222**	25.19	0.761	8.15	**0.184**	24.70	0.835	2.5h
VR	VRGS	×	10.4 ms	7.74	0.286	*26.45*	*0.765*	8.11	0.211	*26.73*	*0.854*	0.4h
	Ours	✓	10.9 ms	*8.02*	*0.237*	26.03	0.755	*8.16*	*0.196*	25.97	0.815	1.4h

Figure 6.7: Example renderings of the PLAYGROUND, FAMILY (upper row), BONSAI, and BICYCLE (lower row) scenes used in the user study.

6.5.3 User Study

To further evaluate our method against the VR-ready Gaussian Splatting VRGS, we conducted a user study. In our study design, we follow Deng et al.'s foveated radiance field user study design and evaluation [Deng et al. 2022]. The experiments were approved by an institutional ethical review board.

Stimuli. Each stimulus set comprised VR-ready stereo rendering using both our method and VRGS. The resolution of each image was 2016×2240 pixels per eye, and seated free exploration was possible. We used scenes GARDEN, BICYCLE and BONSAI from the MipNeRF-360 dataset [Barron et al. 2022] as well as scenes FAMILY and PLAYGROUND from the Tanks&Temples dataset [Knapitsch et al. 2017]. Example renderings are presented in Fig. 6.7 and in the teaser (Fig. 6.1).

Setup. Each participant was seated and equipped with an eye-tracked HTC Vive Pro Eye headset to view stimuli throughout the experiment. The eye tracker was calibrated for each participant with the default calibration kit. The users had a keyboard in front of them to switch rendering modes at will as well as to confirm the choices. Participants could switch the rendering mode by pressing the spacebar and confirm their choice with the enter button. In total, twelve users (aged 22 to 30, 4 female, 8 male) participated in the experiment.

Task. The task followed a two alternative forced choice (2AFC) design. Each trial presented a pair of stimuli generated by our method and VRGS, with the order of viewpoints and the first mode displayed being randomized. A mandatory 0.5 second break [Guenter et al. 2012], displaying a gray screen, was

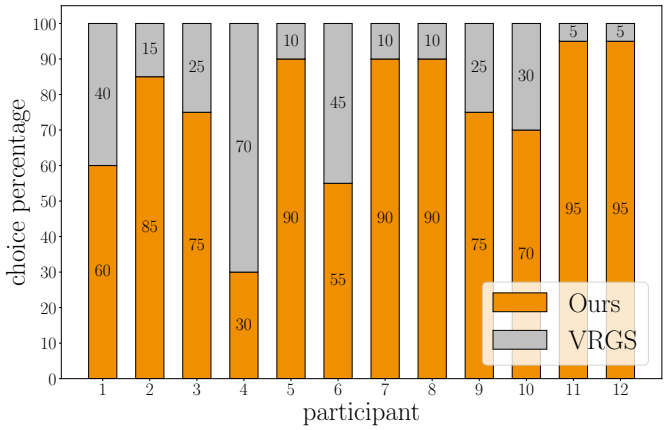


Figure 6.8: Results of our user study per participant, with the majority prefering our method.

introduced between conditions to reset visual perception, with the same applied between different scenes.

Each participant completed 20 trials throughout the experiment. Before the experiment, the participants were briefed and the device was calibrated. During the study, participants were allowed to take as long as needed to complete each test. The participants were unaware of the experimental hypothesis.

Results. Participants in total made 240 decisions, their percentage distributions are shown in Fig. 6.8. We observe a significantly higher ratio of participant who preferring ours to VRGS ($\sim$ 76% preferred ours, $SD = 0.196$). Binomial tests showed significance of this with $p < 0.005$.

In the GARDEN scene, our method was preferred overwhelmingly with over 90%, as our method is able to showcase crisp details accurately (as seen in Fig. 6.1). The PLAYGROUND scene revealed the lowest preference for our method with 67%, as here the initial viewpoint was further away, making details easier to miss for the participants.

Discussion. The high preference ratio for our method confirms the effectiveness of our method. This is underlined by the comments made by the participants, which highlighted the higher perceived quality and sharpness of our renderings.

6.5.4 Ablation Studies

6.5.4.1 Performance Ablation

We ablate our rendering performance in Tab. 6.3, with individual features turned off. Rendering time is evenly split between foveal and peripheral reconstruction (upper part), with peripheral rendering taking 4.9 ms on average.

The enhancements mentioned in Section 6.3 all improve performance in a measurable way (middle part). Opacity penalty for example increases render times by 2.5 ms, while depth culling slightly lowers peripheral render times at the cost of not culling neural points.

Our introduced quality enhancing features (lower parts) have performance impacts, especially the hierarchical sorting approach to fix popping [Radl et al. 2024]. However, as described in Sec. 6.3.1, it proved important to perceptual quality.

	Periphery GS	Fovea Points	CNN	Combine	Tonemapping	Combined
Ours	4.9 ms	3.5 ms	1.6 ms	0.5 ms	0.4 ms	10.9 ms
w/o depth	4.8 ms	4.0 ms	1.6 ms	0.5 ms	0.4 ms	+0.4 ms
w/o smaller neural net	4.9 ms	3.5 ms	1.9 ms	0.5 ms	0.4 ms	+0.3 ms
w/o opacity penalty	7.2 ms	3.7 ms	1.6 ms	0.5 ms	0.4 ms	+2.5 ms
w/o popping fix	2.6 ms	3.5 ms	1.6 ms	0.5 ms	0.4 ms	-2.3 ms
w/o combine e	4.9 ms	3.5 ms	1.6 ms	0.3 ms	0.4 ms	-0.2 ms

Table 6.3: Performance Ablations of our features and improvements. *Periphery GS* refers to the rasterization of the Gaussian splats and *Fovea Points* to the rasterization of neural points.

(a) Ground Truth (b) Ours (0.4M) (c) MCMC (0.5M)

Figure 6.9: Densification strategies. 3DGS-MCMC [Kheradmand et al. 2024] employ an densification scheme up to an exact number of primitives, however their background reconstruction quality is limited compared to ours.

6.5.4.2 Densification Strategy for Gaussian Splatting

Recently, Kheradmand et al. [2024] introduced a Markov chain-dependent Gaussian Splatting densification scheme. It allows for an exact determination of the number of Gaussians compared to our heuristic-based densification. This method would be preferable, as our densification heuristic has to be kept conservative to avoid overshooting the VR limit.

As shown in Fig. 6.9, the low amount of primitives still result in great foreground reconstruction, however, deteriorated backgrounds. This introduces severe artifacts in the peripheral reconstruction, including spiky Gaussians with high color variance, which is unsuitable for smooth peripheral rendering. As such we use the heuristic densification strategy.

6.6 Limitations

We inherit limitations from both 3DGS [Kerbl et al. 2023] and TRIPS (see Chapter 4).

Our approach and the VRGS Gaussian baseline suffer from floaters which is further enhanced by the low densification thresholds used. Passing through floaters can be perceived as flickering and is distracting in VR. Stronger mitigation methods [Huang et al. 2024b; Philip and Deschaintre 2023] would be beneficial in these cases.

The hardware also presents challenges. We had to increase the fovea sizes as the eye tracker was inaccurate during the experiments. Better calibrated and more accurate gaze position allow further performance increases with our method.

Additionally, our approach requires per-scene processing, as such direct virtual teleportation during capturing is impossible and will be looked at in the next chapter.

6.7 Conclusion

In conclusion, the proposed foveated rendering approach demonstrates a viable solution to overcome the computational demand for radiance field rendering in real-time virtual reality systems. By blending high-detail neural point rendering for the foveal region with smooth 3DGS rendering in the periphery, we obtain crisp rendering of radiance fields without compromising system performance. The hybrid solution meets the performance demands of interactive VR, offering a significant improvement in rendering quality and user experience, which might pave the way for more realistic and immersive VR applications such as video games or virtual tourism.

CHAPTER 7
Inovis: Instant Novel-View Synthesis

This chapter is based on the publication Harrer and Franke et al. [2023]. The author of this thesis contributed the project idea, algorithmic idea, and supervised Mathias Harrer during the implementation in their Master's thesis. In addition, the author of this thesis contributed in part to generating results for related methods and wrote the original draft of the paper. Laura Fink also contributed parts of related method evaluations. Mathias Harrer contributed the implementation and major parts of the method evaluation and validation. Marc Stamminger and Tim Weyrich administered, helped conceptualize and supervise the project as well as provided critical feedback in all stages. All co-authors contributed review and editing during writing.

7.1 Introduction

While the previous chapters introduced systems for high-quality, immersive scene exploration, they require the capturing process to be completed before per-scene processing is done. This means the delay from capturing to exploration ranges from minutes to hours. For immersive virtual teleportation, however, it would be preferable to be able to explore virtual environments of initially incomplete scene, during capturing, and without the requirement of preprocessing the scene beforehand.

This approach seeks to work with dynamic input—and an evolving model—at render time, assuming a SLAM-like setting in which a (point-based) 3D representation is gradually built up.

Our rendering system's main advantage is its use of original frames near the target view, integrating them with the model rendering for enhanced fidelity, instead of solely depending on the fused model. These nearby frames, termed *auxiliary frames*, provide detailed information where a high-resolution model is incomplete. Utilizing temporal supersampling techniques [Xiao et al. 2020], recombination and fusion with a direct neural rendering of the point-based representation are enabled in our framework.

In effect, our method is designed to use fewer resources and capturing constraints: new scenes do not require new training (unlike Aliev et al. [2020], Rückert et al. [2022], or TRIPS (see Chapter 4)) and the method works directly on sparse point clouds (unlike Stable View Synthesis [Riegler and Koltun 2021],

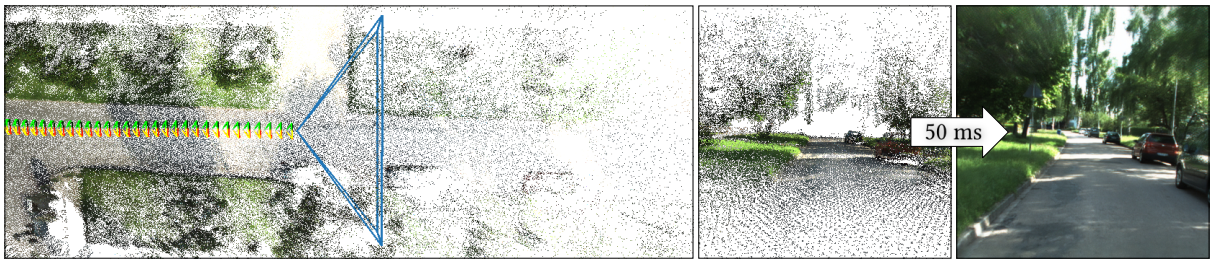


Figure 7.1: Our neural point-based rendering architecture is particularly suited for scenarious where novel, previously unseen content is dynamically added to a scene. *Left:* simulation of a gradually built up LiDAR-based reconstruction; top-down view of the novel views' frustum and frustums of past frames, which are used as input. Although the scene ahead of the camera is increasingly sparse with distance (*center*), our architecture manages to render a novel view with remarkable fidelty (*right*).

which requires highest-quality geometry). Thus, our method enables real-time NVS on dynamic LiDAR or depth-map streams, which was not achievable with neural point renderings before. In summary, our contributions are:

- An instantaneous, high performance method for NVS that compares favorably to the state-of-the-art in similarly constrained scenarios.
- NVS in previously difficult or impossible scenarios, such as live depth map streams.
- An optional temporal feedback loop that reuses previously rendered novel views to augment the set of RGBD captured images for improved temporal coherence.

The open-source implementation of the method can be found at:

`https://github.com/mharrer97/inovis/`

The project page can be found in this repository and at:

`https://reality.tf.fau.de/publications/2023/harrerfranke2023inovis/`

7.2 Related Work

Neural Novel View Synthesis. For point-based representations and our reduced preprocessing domain, closest to our method is NPBG++ [Rakhimov et al. 2022], which integrates features from all input images (via a generalized feature encoder) within a point cloud that is then neurally interpolated to render novel views. It involves a few minutes of preprocessing and renders novel views in real time. In contrast to NPBG++, however, our method does not require preprocessing and is not bound to a priori scene completeness; moreover, its lower per-point memory footprint improves scalability to larger scenes.

Other neural IBR-based methods [Riegler and Koltun 2021; Wang et al. 2021] present challenging inference speeds and generally depend on a priori scene completeness, prohibiting use on live camera streams.

Recent work that bridges implicit and explicit approaches tackles the task of online scene reconstruction [Clark 2022; Müller et al. 2022; Sucar et al. 2021]. These involve training a reconstruction and creating respective novel views during capture time; however, time to image (training plus rendering) is still in the order of seconds for high-fidelity images. Furthermore, faithful reconstructions require repeated observations of the same objects.

This method tackles these shortcomings, supporting incomplete scene observations, generalized feature encoding without scene-specific pre-training, and fast inference without online training.

Neural Supersampling. A specific instance of NVS involves real-time (spatio-temporal) upsampling of low-resolution inputs, widely used in video upsampling [Kappeler et al. 2016; Liu and Sun 2013; Tao et al. 2017] and more recently in video games via methods like Nvidia's DLSS [Edelsten et al. 2019]. Modern techniques often employ neural networks to enhance the input stream [Liu et al. 2022]. Some solutions focus on temporal supersampling by projecting temporal data onto current frames [Guo et al. 2021; Thomas et al. 2022; Xiao et al. 2020]. Our key finding is that upsampling from coarse data and NVS from sparse point clouds share a common goal—completing missing information—and both utilize feature encoding and interpolation. This paper leverages prior work on temporal supersampling, particularly Xiao et al. [2020]'s neural supersampling feature encoding, to develop a rapid, lightweight novel-view synthesis pipeline that effectively corrects common warping artifacts.

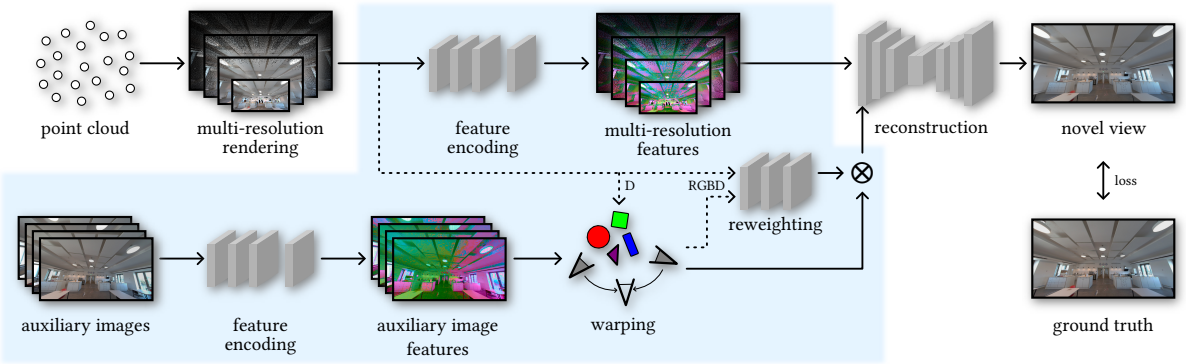


Figure 7.2: Our rendering pipeline extends the approach of neural point rendering of Aliev et al. [2020]; novel components are highlighted in blue. From a set of auxiliary (RGBD) images from nearby input (or previously rendered) views, relevant views are selected and encoded. Guided by a rendering of the point cloud, they are warped and reweighted to mitigate occlusion and ghosting effects. Subsequently, features from the multi-resolution rendering of the point cloud are extracted, and these buffers are passed to the reconstruction network for the final rendering. The pipeline is trained end-to-end against ground truth images.

7.3 Method

As in the previous chapters, we employ a neural point method. In their original design, appearance descriptors are precomputed in an off-line process, and the multi-resolution renderings to be fed into the U-Net are rendered in feature space. In contrast, our approach seeks to work with dynamic input—and an evolving model—at render time. Accordingly, and with the convenient side-effect of memory savings that allow for larger scenes, point and image attributes are kept in the original RGB(D) format for longer, and feature vectors are extracted only on the fly, see our pipeline overview in Fig. 7.2. (The encoder is pre-trained using a class of similar input scenes.)

In general, we assume a SLAM-like setting, in which a (in our case point-based) 3D representation is gradually built up while additional RGBD data are progressively received and fused into the model.

As a key contribution of our method, our rendering system does not simply rely on the fused model representation, but also capitalizes on the availability of original frames close to the target view to be rendered, by combining the model rendering with a more image-based approach that recombines these nearby frames to fill in higher-fidelity information where a consistent high-resolution model is not yet available. Notably, these additional frames (we call them *auxiliary views*) may contain only sparse depth data (as, e.g., in the case of LiDAR sensors) but are assumed to be dense in RGB.

Although similar in spirit to traditional IBR and texturing [Buehler et al. 2001; Schönberger et al. 2016], we blend feature descriptors rather than RGB values and borrow architectural aspects from the design of Xiao et al. [2020].

In concrete terms, for every output (target) frame, our system determines nearby original camera images (Sec. 7.3.2), converts them into feature space (Sec. 7.3.3) and warps them into that target frame (Sec. 7.3.4). The warp is assisted by depths of the so far accumulated 3D model. The different source pixels are blended using a reweighting network [Xiao et al. 2020], which leads to better blending results over explicit weighting functions and hole-filling strategies (Sect. 7.3.5). Subsequently, the features of the multiresolution rendering of the point cloud are extracted (Sec. 7.3.1) and combined with the blended (feature) images before being passed to the reconstruction network for rendering (Sec. 7.3.6).

Any (supervised) pre-training takes place using this complete pipeline (Sec. 7.4). In Sec. 7.5 we test the system both with previously observed data (in order to allow comparison to previous works) and with dynamic input outside the training dataset, a scenario not normally considered by previous work.

7.3.1 Target View Feature Extraction

The backbone of our rendering architecture is a neural point-based renderer that roughly divides into two parts: sparse multiresolution encoding of a target frame (which we discuss here); and U-Net-based neural rendering that transforms this representation into an RGB rendering (Sec. 7.3.6).

First, we render the raw (RGB) point cloud with a depth-testing, one-pixel per point OpenGL hardware renderer to obtain a sparse RGBD image. In a second step, we then use a feature encoder with the same structure as the auxiliary view's feature extractor (Sec. 7.3.3), except a filter number of 32 and output of 8 features derived from the rendered sparse point cloud. In the output, the resulting features are concatenated again with the network input.

These two steps are repeated for every resolution level fed into the multiscale rendering U-Net described (Sec. 7.3.6). In our experiments we use three lower-resolution point renderings, at $1/2$, $1/4$, and $1/8$.

The multiresolution structure serves two purposes: more general occlusion features can be learned from these coarser representations; during training, the feature extractor sees a more varying range of point distributions, making the feature extractor more robust.

7.3.2 Auxiliary View Selection

We collect n_{views} *auxiliary images*, taken from nearby captured RGB images. Depending on the sensor types and acquisition system, captured depth images are used (RGBD scenes), or (sparse) corresponding depth samples are generated from the LiDAR data using the same point renderer we use in Sec. 7.3.1. Either depth representation is used without any further preprocessing.

We generally determine the n_{views} nearest views by selecting those that minimize the following metric based on a positional factor f_p and a view direction factor f_d:

$$f_p = 0.5 + \max\left(\|\vec{p} - \vec{p}_{\text{target}}\|^2, 0.5\right) , \tag{7.1}$$

$$f_d = 1 - \left(\frac{\vec{v}}{\|\vec{v}\|}\right)^{\top} \cdot \frac{\vec{v}_{\text{target}}}{\|\vec{v}_{\text{target}}\|} . \tag{7.2}$$

Here, p_{target} and v_{target} are the position and view direction of the target view, and p and v are the position of the auxiliary buffers (viewpoint) and view direction. In general, positions close to the target position will always contribute more, thus scoring lower—as do similar view directions. Combining these two factors then results in a similarity score s, where lower values indicate better view choices:

$$s = f_p \left(1 + \alpha f_d\right) . \tag{7.3}$$

Hereby, α is used to balance these two factors, with larger α prioritizing tighter view directions. We use $\alpha = 100$ for a trade-off that penalizes distances of around 10 meters similarly to 90-degree view offsets. Views are always ordered from best to worst, which leads the network to place more emphasis on the closest view. The resulting view weighting and ranking in spirit resembles traditional view-dependent rendering approaches [Buehler et al. 2001]; More complex view selection schemes, such as ORB-SLAM [Campos et al. 2021], could be used but were deemed to offer diminishing returns within the scenarios at hand.

For some datasets, varying quality of nearby views could lead to moderate flickering in the output video. In these cases, adding the previously *rendered* view to the set of candidates for auxiliary views

point rendering | no environment encoding | environment encoding | closest ground truth

Figure 7.3: Comparison of environment encoding vs no environment encoding vs point rendering vs closest ground truth. Environment encoding significantly improves image quality in areas unoccupied by points. Even without environment encoding, the surrounding areas of points are filled with information, showing the capability of encoding a patch around a pixel into a feature tensor which is then warped, using a sparse point rendering.

helps mitigate flicker: that view often scores highly on the nearby-view metric and, if included amongst the n_{views} frames, improves temporal coherence across output frames. For training, however, previous frames are never considered; our pipeline is robust enough to easily handle a previously rendered frame as input, considering that it is fully compatible with the original RGBD-captured frames.

7.3.3 Feature Encoding for Auxiliary Views

After view selection, a lightweight feature encoding network transforms RGBD pixels into 12-dimensional feature vectors. Note that the encoder captures spatial context as well, so that even under sparse (point-wise) reprojections, as applied in the following section, relevant spatial information is preserved.

We use a three-stage gated convolution setup with ReLU activations and a filter number of 16. (Xiao et al. [2020] use 32; our experiments showed no change in training loss when going from 16 to 32, so we went with fewer weights.)

Also in contrast to Xiao et al., we use gated convolutions [Yu et al. 2019], which have learnable masks for scaling inputs. As our auxiliary images' depth maps are sparsely filled, passing this map to gated convolutions allows the feature encoding network to identify areas which can not be correctly warped (due to missing depth) and to amplify this information to adjacent, warpable pixels. Thus, gated convolutions in our case allow for a form of "inverse" masking, where information warping is able to account for sparse depth maps similarly to small-scale attention mechanisms in transformers [Vaswani et al. 2017]. Lastly, this allows us to use a smaller number of filters, which generally improves performance and reduces the risk of overfitting.

7.3.4 Warping

For each auxiliary view, both features and RGBD are warped to the target view: for each point of the point cloud visible from the target perspective, its pixel value (RGBD plus feature vector), at its back-projection into the auxiliary view, is copied to the new location. Each auxiliary image is warped separately and the resulting images are mixed together in later pipeline steps.

Pixels that do not feature rasterized points miss warped information. These areas are optionally augmented by warping the auxiliary image onto a distant background plane behind the point proxy geometry. This helps to greatly improve image quality in areas that do not feature points, especially for

far away areas (e.g., sky or far-away buildings). This environment handling results in those areas being filled with plausible content from the background, as the disparity is barely noticeable at these large distances, see Fig. 7.3.

Then, the RGBD images from the auxiliary views and the original rendering are passed to the reweighting network, which computes a pixel-wise weighting factor for each view to scale the features.

7.3.5 Reweighting

This step helps to lessen the effect of disocclusion and warping artifacts. Reweighting mostly follows the work of Xiao et al. [2020]. The reweighting network takes RGBD information from both warped auxiliary views and the rendered target view to determine per-pixel weights for all n_{views} auxiliary images as output. The warped image data is weighted accordingly by multiplying the weights with the auxiliary features. The reweighted data is concatenated and passed to the front layer of the neural render network.

The network consists of three blocks of convolution and ReLU with filter numbers of 32, outputting n_{views} multiplication factors, scaled to $[0, 10]$ as stated in the original paper [Xiao et al. 2020].

7.3.6 Neural Render Network

The neural rendering network outputs the final stable novel view of the requested extrinsics and intrinsics. It is a U-Net with five levels, similar to established point-cloud rendering inpainting networks [Aliev et al. 2020; Rückert et al. 2022]. Each downsampling block uses a gated convolution and max-pooling layer, while each upsampling block uses bilinear upsampling and a gated convolution with skip connections between the blocks. Our filters are 32, 32, 32, 32, 32, which provided the best trade-off between quality and performance and converge faster than deeper networks.

We input the full-resolution encoded features of the target and auxiliary views to the network, with the lower-resolution target feature maps progressively being added to their respective downsampling blocks.

7.4 Training Methodology

Scenes Tab. 7.1 summarises the scenes used for training (and evaluation), which cover a wide range of scenarios and setups, including depth capturing through LiDAR, MVS, and RGBD cameras; various indoor and outdoor environments, as well as inside-out and outside-in capturing setups.

Table 7.1: Evaluation datasets. We use a wide range of possible scenarios to evaluate our approach. Scene tagged with *live* indicate an expanding live dataset.

Dataset	# Points per scene	Point Cloud capturing	# Image per scene	Image Resolution	Capturing Methodology	Environment Type
Tanks & Temples	~10M	MVS	~300	$1920 - 2144 \times 1088$	outside-in	various
Kitti-360	~8M	**Live**-LiDAR	~630	1408×352	sequential	outdoor driving
ScanNet	~30M	**Live**-DepthCam	~120	1280×960	inside-out	indoor rooms
Redwood	~35M	**Live**-DepthCam	~150	640×480	outside-in	object scans
Office	~70M	LiDAR	~700	960×544	inside-out	indoor rooms
Custom	0.5M-40M	**Live**-StereoCam	40-250	$1280 - 1920 \times 720 - 1080$	sequential	outdoor scans

We use the *Playground* and *M60* scenes from the commonly used Tanks and Temples dataset [Knapitsch et al. 2017], which has depths estimated with MVS. Several sequences from *Kitti-360* [Liao et al. 2022b] offer sequential recordings of a driving car captured with LiDAR, which we modified to simulate the order in which depth samples and RGB images would come in within a live LiDAR system (based on field of view and distance of the original Kitti 360 point cloud). Additionally, we used our own captured scene, *Office*, using a portable indoor LiDAR, with capture positions roughly 2m apart. Extensive preprocessing yielded a cleaned-up point cloud with a resolution of 5mm.

For live-RGBD scenes, we use the ScanNet [Dai et al. 2017] and Redwood [Choi et al. 2016] scenes, which are captured RGBD streams for which we estimated positions on the fly with ORB-SLAM [Campos et al. 2021] where no poses where present. Apart from that, we captured *custom outdoor scenes* using a stereo camera, featuring sequential traversal of environments without repeated observation of the same objects. We obtained the poses and keyframes for these scenes with Snake-SLAM [Rückert and Stamminger 2021]. For all live-RGBD scenes, we create timestamped point clouds from the last 10 images in the sequence.

Training Training is generally initialized with a network pretrained on the Office scene, owing to its large spatial extent combined with large baselines that prime training against ghosting. Subsequently, networks are refined for individual classes of scenes, without temporal feedback frames (using captured data only). Testing, naturally, always takes place on previously unseen data.

Loss As training loss, we use a mix of VGG16 [Johnson et al. 2016] and SSIM [Wang et al. 2004]. The VGG loss is a common feature-based function, which is especially well suited in our case (as seen in Sec. 7.5.5). However, regular patterns can appear on surfaces, which can be removed by adding a small amount of SSIM loss. Our final loss function is the following ($w_{\text{ssim}} = 0.25$ and $w_{\text{vgg}} = 1$):

$$\text{loss}(\tilde{x}, x) = w_{\text{ssim}} \cdot (1 - \text{SSIM}(\tilde{x}, x)) + w_{\text{vgg}} \cdot \text{VGG16}(\tilde{x}, x) \tag{7.4}$$

Inference Our method is particularly effective when applied to live datasets that supply a steady stream of new images, such as Kitti-360, Redwood, ScanNet, and our custom outdoor scenes. The images are encoded during inference and selected based on their capture position and rotation, enabling us to choose images without having to store them in memory. This makes our approach well-suited for integration with a traditional streaming approach, which only holds a subset of images in memory, thereby minimizing memory usage.

Additionally, we only store RGB data in our point cloud, compared to competing methods such as NPBG++ [Rakhimov et al. 2022], who store large multi-channel descriptors per point. All in all, our rendering is lightweight and is easily interactive (~50ms/frame rendering time on Kitti and Redwood datasets).

7.5 Results

In the following, we present our evaluation of Inovis compared to related work, as well as ablation studies to give insights on what makes our method effective. For qualitative results, see Fig. 7.4.

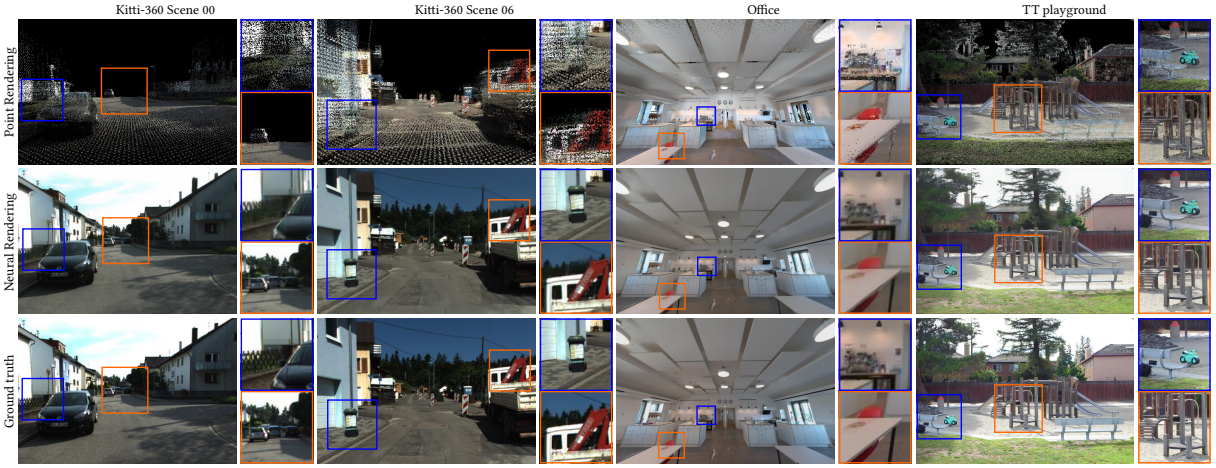


Figure 7.4: Neural renderings of our approach for three datasets: the Office dataset and the Tanks and Temples dataset, both containing high quality point clouds and the Live LiDAR dataset Kitti-360. Our method produces high quality neural renderings for all datasets.

Table 7.2: Comparison with SVS, IBRnet, ADOP and NPBG++. Except for ADOP, no scene-specific finetuning is performed.

	Motorcycle			Sofa			ScanNet		
	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑	PSNR↑	LPIPS↓	SSIM↑
ADOP	20.34	0.236	0.585	24.26	0.172	0.598	24.78	0.208	0.694
IBR-Net	22.93	0.193	0.730	27.45	0.159	0.809	24.96	0.248	0.802
NPBG++	14.70	0.543	0.497	17.02	0.392	0.637	20.90	0.396	0.758
SVS	19.21	0.273	0.658	22.35	0.253	0.728	23.76	0.316	0.796
Inovis (ours)	20.63	0.195	0.698	24.19	0.187	0.769	22.78	0.309	0.792

7.5.1 Qualitative Evaluation

We compare our approach to established neural novel view synthesis techniques. These include *Stable View Synthesis* (SVS) [Riegler and Koltun 2021], NPBG++ [Rakhimov et al. 2022], IBR-net [Wang et al. 2021] and ADOP [Rückert et al. 2022].

For our method, a subset of scenes is used to train a network tuned to one scene type. Baseline methods that have generalizing capabilities (all except ADOP) were trained on the whole Redwood and ScanNet datasets, except the unseen evaluation scenes. ADOP was individually trained on each evaluation scene to allow for a comparison. We accumulate metrics over five unseen scenes from ScanNet (0, 20, 30, 40, 50, 80), five Redwood motorcycles (05489, 05751, 05984, 06186, 06190) and three Redwood sofas (00577, 05477, 07294). The results of this evaluation can be seen in Tab. 7.2, Tab. 7.3 and Fig. 7.5.

In terms of quality, we consistently outperform NPBG++, while they show slightly better render times. SVS shows similar results with significantly slower rendering. ADOP and IBR-Net usually provide better metrics, but neither method in its current form is suitable for online novel view synthesis: ADOP requires scene-specific training (~6h) and IBR-Net shows long rendering times for single images (~2 min). In general, our approach provides state-of-the-art results with fewer limitations than competing approaches.

Table 7.3: Approximate preprocessing and render times (ScanNet 1296×968).

	preprocess	rendering
ADOP	~ 6 h	~ 13 ms
IBR-Net	< 1 min	~ 2 min
NPBG++	~ 1 min	~ 100 ms
SVS	~ 4 h	~ 3 s
Inovis (ours)	-	~ 131 ms

Table 7.4: Performance numbers in ms: inference (Inf), point rendering (PR) and total rendering time (T) on single scenes.

Dataset	Inf	PR	T
Kitti (3.6M pts, 1408×376)	53.8	1.2	56.2
ScanNet (70M pts, 1296×968)	120.6	9.7	131.7
Redwood (36M pts, 640×480)	36.7	9.3	47.4
Office (40M pts, 960×540)	58.9	12.9	73.1
Custom (0.6M pts, 1280×720)	88.3	0.6	90.2

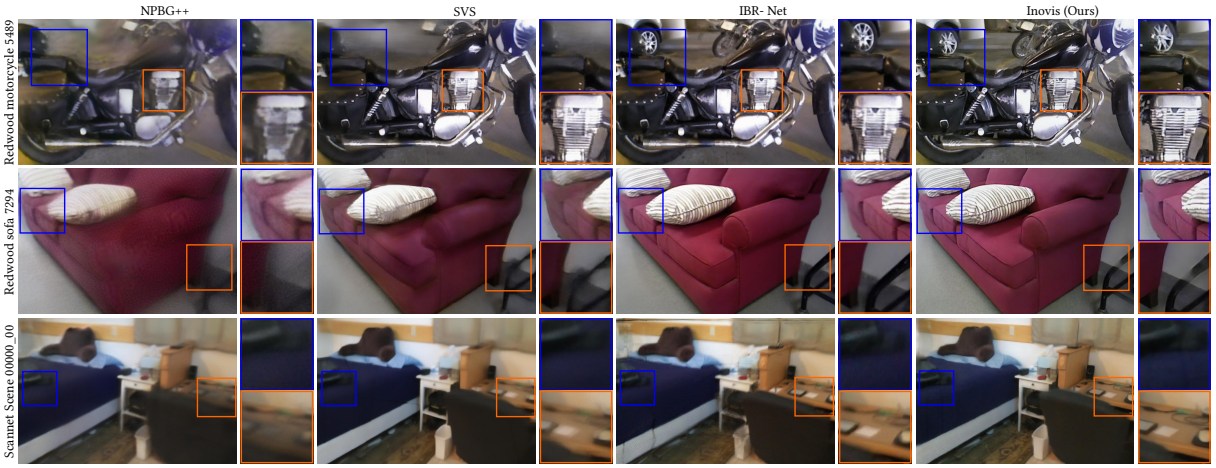


Figure 7.5: Visual comparison of Inovis with NPBG++, SVS and IBR-Net. Our method produces visually similar results to IBR-Net, while outperforming NPBG++ and SVS.

7.5.2 Generalization

To further demonstrate generalization, we used a subset of scenes from the Kitti dataset to train a network that was also used to create all evaluation images of our own custom dataset.

The results can be seen in Fig. 7.6. Each method is provided with different time budgets that match the data capture time frame of the respective dataset, e.g., a time budget of 15s for creating a novel view from 15 keyframes. (*column 1–3*). The last *column* contains the results for unlimited time budgets. Both used datasets resemble a sequential trajectory through a scene, rather than continuous observation of a single object: walking past an excavator (Custom scene; *top*); a car driving through a neighborhood (Kitti-360; *bottom*). Training and rendering times, which result in total time to image by adding up, are displayed in each image. For all examples shown, our system was pre-trained on a subset of Kitti-360 scenes, excluding the one on display. Without further training, our network generalizes well to the shown scenes, thus limiting the total time to image to mere rendering time (~50–100ms) and resulting in the exact same image, no matter the time budget. ADOP and Instant-NGP gradually improve with increasing time budget but do not reach our quality as long as a time limit is in place.

While these are also outdoor scenes, their characteristics are different, further corroborating generalizing power.

7.5.3 Performance Evaluation

Training. Training our network end-to-end takes around ~15–24h on an NVIDIA V100, depending on dataset size and resolution.

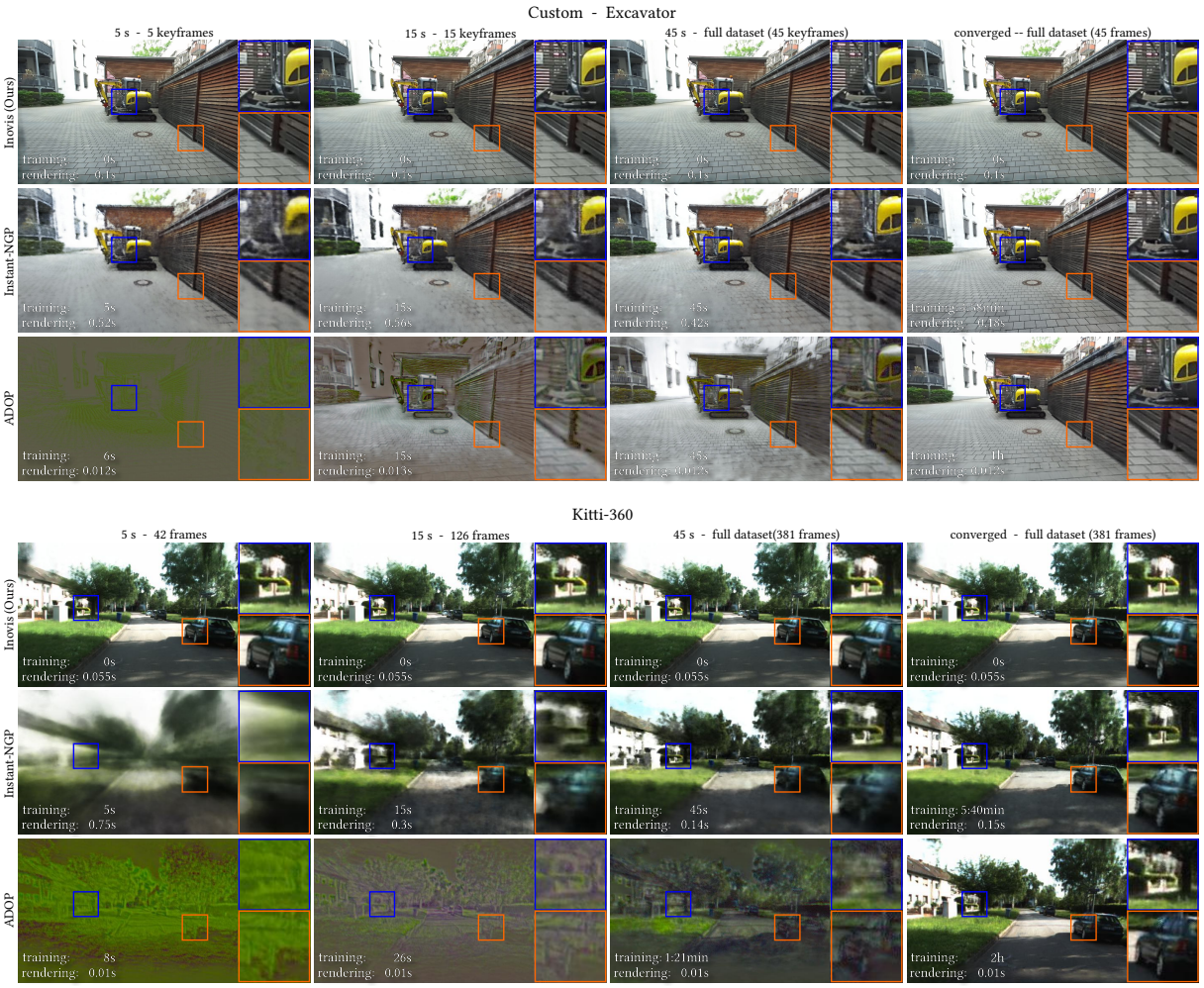


Figure 7.6: Comparison of streaming capabilities between ADOP, Instant-NGP and Inovis (Ours).

Rendering. Training is done once per scene type, hence runtime performance depends on rendering (i.e., NVS) only, which highly depends on the number of auxiliary views (cf. Sec. 7.5.5) and resolution. See Tab. 7.4 for an overview of rendering times for different datasets, obtained with an NVIDIA A5000. We measured interactive frame times of up to 20 fps for 960×540 images and ~11 fps for 1280×720 images. Even for large point clouds (tens of millions of points), inference still dominates rendering time, but differences in visible point count cause variations in total rendering time, as illustrated in Fig. 7.7.

To assess potential for streaming scenarios, we compare our approach against two competing methods: ADOP [Rückert et al. 2022], as a high-quality point-based approach with very fast neural rendering; InstantNGP [Müller et al. 2022], which features remarkable real-time training of a NeRF, especially when applied on outside-in scenarios. Considering that neither ADOP's nor InstantNGP's publicly available implementations were designed to incrementally ingest streamed content, we devised an evaluation regime in their favor that, for a selection of time stamps within a stream of keyframes, grants each algorithm combined training and rendering time roughly equivalent to the time passed until that timestamp; see Fig. 7.6 for the results. Our approach outperforms both methods, as no training is needed and produces high fidelity images from the start, while both compared methods only gradually improve visual quality with increasing time budget. We outperform both methods in all comparisons

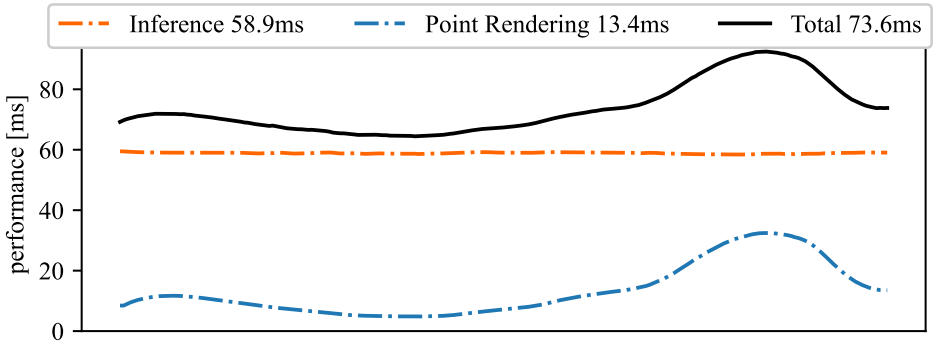


Figure 7.7: Performance over time during traversal of the office scene (960×540, 4 auxiliary views), featuring a large point cloud (40M points). Due to the large point count, point rendering time is significant. Inference time remains constant while rendering time varies with the number of visible points.

w.r.t. time-to-image (training plus rendering) and achieve our highest quality even within the minimum time budget.

7.5.4 Use-Case Evaluation

Our technique adeptly addresses various problem categories often overlooked by NVS methods.

Live LiDAR Car Data. For driving scenarios, refer to Fig. 7.4 and Fig. 7.6. Utilizing the Kitti-360 car dataset, where new content frequently appears in front of the camera, we demonstrate our method's proficiency in generating views in areas with limited data. If the captured data belongs to a similar category as the scene our model is trained on, we can swiftly generate novel views from a minimal set of images and the current point cloud.

Large Point Clouds. Neural methods often struggle with large point clouds. Conversely, our approach effectively manages these large datasets, exemplified by the Office dataset with 75M points, as depicted in Fig. 7.4.

Sparse Point Clouds. Our method enhances sparse point data with image details by encoding pixel surroundings into feature vectors, as demonstrated with the sparse LiDAR data of the Kitti-360 dataset. See Figs. 7.3, 7.4, and 7.6.

Live RGBD Data. Our approach is well-suited for live RGBD data. Given real-time camera poses from SLAM, and a pre-trained network for a similar scene, Inovis generates new images without scene-specific training.

7.5.5 Ablation Studies

In this section, we provide studies to show the root of the effectiveness of our method. Variants were evaluated by using our loss function (Sec. 7.4) on the hold-out set of captured frames.

Table 7.5: Loss by convolution type of feature encoding of auxiliary images, by dataset.

Dataset	basic conv.	gated conv.
Office	0.1867	0.1798
Kitti-360	0.5807	0.5561

Table 7.6: Loss functions on the Office dataset. *NSS* indicates the loss function used by Xiao et al. [2020].

Loss Function	PSNR↑	LPIPS↓	SSIM↑
Train w. MSE	24.14	0.141	0.891
Train w. SSIM	24.16	0.110	0.917
Train w. VGG	23.85	**0.090**	0.913
Train w. NSS	**24.27**	0.102	**0.919**
Train w. Ours	24.22	0.093	0.917

Feature Extraction. Our pipeline encodes small patches of auxiliary view information into feature vectors, allowing this information to be transferred to novel views, as seen in Fig. 7.3. As described in Sec. 7.3, we use gated convolutions, which usually provide state-of-the-art hole-filling capabilities. In contrast to that, we use their masking abilities inversely for auxiliary image encoding, with the sparse depth map guiding information amplification. For results of this, see Tab. 7.5. In sparse point cloud scenarios (such as LiDAR in cars) this addition provides great improvement in the relevant (usually small) areas, thus increasing training convergence and providing improvements of up to 5% overall.

Number of Auxiliary Images. We use a comparatively small number of auxiliary views compared to competing methods, which for our setup suffices. As seen in Fig. 7.8, using more auxiliary images shows diminishing results especially considering the increased computation time per frame, which increases linear with number of images used. Four seems the best tradeoff between quality and inference speed. In our experience, using four images also helps to improve quality when deviating from ground truth views, although two images already show a similar loss compared to four images.

Loss function. We compare different loss functions commonly used in novel view synthesis methods in our method. For results, see Tab. 7.6. Numerically, our metric performs favorably, only slightly surpassed by the loss function of Neural Supersampling [Xiao et al. 2020]; subjectively, however, we find that our metric provides higher color accuracy for our datasets, suggesting that real-world datasets require differing loss compositions than synthetic data.

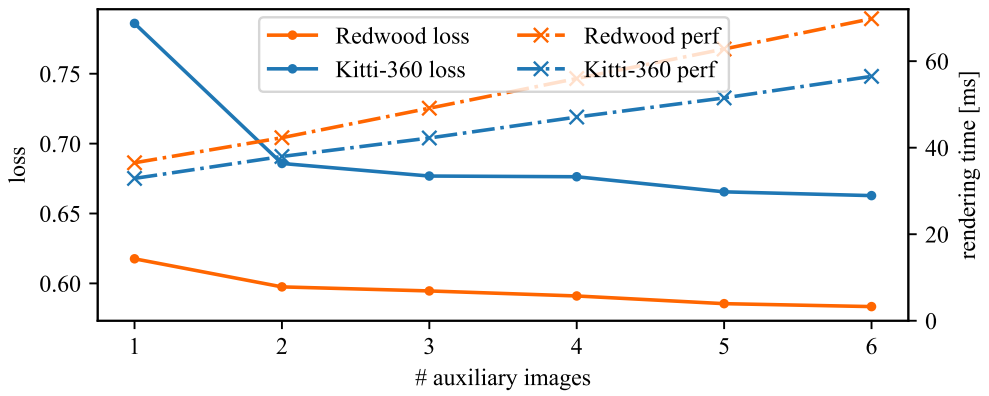


Figure 7.8: Loss and rendering performance vs auxiliary images used.

7.6 Discussion

Input Quality. Our method relies on good quality captures to produce high-quality results. This includes the capture positions of the images not be too far apart in order for the method to work effectively. In general, regions without overlap from auxiliary images show poor quality due to missing information to be interpolated. We fall back on averaged pixel colors in our point cloud rendering, but these can exhibit artifacts.

Temporal Stability While our method is able to handle continuous streams of new images, it may still struggle with scenes that have significant changes over time. For example, significant changes of the auxiliary images result in visible flickering.

Camera Intrinsics Our pipeline is trained to produce output images with similar camera intrinsics and resolutions as the input images. Thus, if encoded patches are interpreted with different camera characteristics, the output image can be blurred or squished.

7.7 Conclusion

In this chapter, we presented a system for neural NVS. We adapted a supersampling architecture, which resamples previously rendered frames, to instead recombine nearby camera images in a multi-view dataset. The resulting architecture gains sufficient robustness to significantly improve transferability to previously unseen datasets. In particular, our system enables novel applications for neural rendering where dynamically streamed content is directly incorporated in a (neural) image-based reconstruction of a scene. We show that our method reaches state-of-the-art performance when compared to previous works that rely on static scenes; in addition, we demonstrated our system's performance for dynamically streamed content, a scenario not accessible to previous works in neural rendering.

CHAPTER 8
Conclusion & Prospect

In this thesis, we have explored novel techniques for real-time, high-fidelity scene reconstruction and rendering, with a particular emphasis on VR-ready applications. The overarching goal was to enable seamless virtual exploration and teleportation into real-world environments by addressing fundamental challenges in NVS, including geometric inconsistencies, scalability, and perceptual quality. Additionally, rendering performance must ensure low-latency real-time feedback while maintaining smooth motion, proper depth perception, and perceptual continuity. To tackle these challenges, we have explored novel techniques for real-time, high-fidelity scene reconstruction and rendering, leveraging a point-based representation integrated with differentiable rendering and neural refinement techniques to ensure both efficiency and high-quality results.

We first tackled the problem of geometric deficiencies in point-cloud-based NVS. Chapter 3 introduced Visual Error Tomography (VET), a novel approach for detecting and correcting missing or erroneous geometry directly from 2D visual error maps. By lifting these errors into 3D space, our method allows for targeted refinement, significantly improving rendering accuracy and temporal stability. Unlike traditional point-growing techniques, VET is particularly effective at handling large missing regions and thin structures, reducing artifacts that disrupt user immersion in virtual environments.

Building on this foundation, Chapter 4 presented Trilinear Point Splatting (TRIPS), a hybrid rendering pipeline that merges principles from Gaussian splatting and neural point rendering. By rasterizing points into a screen-space image pyramid and reconstructing missing detail with a lightweight neural network, TRIPS achieves superior rendering quality at real-time frame rates. Importantly, this pipeline remains fully differentiable, enabling automatic optimization of both point positions and sizes, further enhancing robustness.

To address scalability constraints, we introduced Neural Point Octrees (NePO) in Chapter 5. This method organizes massive point clouds into an octree structure, allowing adaptive Level-of-Detail selection during rendering. NePO enables real-time rendering of vast, city-scale environments while maintaining high fidelity. This advancement removes the previous limitations of point-based neural rendering, which traditionally focused on object-centric or room-scale scenes.

Beyond technical advancements, we also investigated perceptual optimizations tailored for VR rendering. Chapter 6 proposed VR-Splatting, a foveated rendering approach that leverages the characteristics of the human visual system. By using smooth Gaussian representations for peripheral vision and high-resolution neural point splatting for the fovea, our method maximizes rendering efficiency without sacrificing perceptual quality. A user study confirmed that this hybrid approach enhances perceived sharpness and realism while maintaining VR's stringent performance requirements.

Finally, Chapter 7 introduced Inovis, a technique for immediate novel-view synthesis using dynamically streamed content. This method eliminates the need for extensive preprocessing, enabling virtual teleportation with minimal delay. By leveraging neural spatio-temporal supersampling, the approach reconstructs missing scene details on-the-fly, making interactive scene exploration more fluid and accessible.

In summary, this thesis has introduced a series of novel contributions that collectively advance the state of the art in neural point-based rendering. By addressing critical challenges in geometric consistency,

scalability, perceptual adaptation, and real-time rendering, our methods pave the way for more immersive and efficient virtual scene exploration. These contributions enable users to explore vast real-world environments realistically in VR, AR, or desktop applications.

One promising avenue for future research is the extension of our methods to dynamic scenes. Currently, the primary focus is on static environments, often masking out or ignoring dynamic elements such as people or moving objects. Incorporating temporal coherence into our framework would allow for more realistic and immersive experiences, enabling virtual scenes to adapt to time-varying elements. This could be achieved through motion-aware neural fields or dynamic point-based representations that track and render moving objects seamlessly. Additionally, integrating learned priors for dynamic behaviors could enhance scene interactions, allowing for responsive and interactive virtual environments.

Another key direction is enhancing the controllability of point-based scene representations for artistic and interactive applications, such as video games and virtual production. Novel representations, while powerful for NVS, are often difficult to integrate into traditional creative workflows. Future work should explore hybrid models that allow direct editing of scene elements, providing artists with intuitive tools to manipulate lighting, textures, and object placement while retaining the benefits of neural rendering. Additionally, optimizing physical properties such as material reflectance and light transport would further improve realism, allowing neural point-based methods to better approximate real-world physics for applications in photorealistic rendering and scientific visualization.

Furthermore, advances in neural point representations and optimization could enable efficient content generation. By integrating large language diffusion models or GANs with neural point-based representations, it may be possible to automate the creation of complex environments from, e.g. textual descriptions, significantly streamlining content generation.

Lastly, an important, yet often overlooked aspect is power optimization for both mobile device compatibility and environmental sustainability. Current high-fidelity neural rendering methods require substantial computational resources both for training and rendering, limiting their feasibility for energy-constrained devices, such as mobile VR headsets or AR glasses. Future research should focus on energy-efficient optimization and hardware architectures that minimize power consumption while maintaining quality. Furthermore, optimizing rendering pipelines for green computing could reduce the environmental impact of large-scale neural rendering, making it a more sustainable technology for widespread adoption.

Bibliography

Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. 2017. Latency Requirements for Foveated Rendering in Virtual Reality. *ACM Transactions on Applied Perception (TAP)* 14, 4 (2017), 25.

Marc Alexa, Markus Gross, Mark Pauly, Hanspeter Pfister, Marc Stamminger, and Matthias Zwicker. 2004. Point-based computer graphics. In *ACM SIGGRAPH 2004 Course Notes* (Los Angeles, CA) *(SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 7–es.

Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. 2020. Neural Point-Based Graphics. In *European Conference on Computer Vision (ECCV)*. Springer, Springer International Publishing, Cham, 696–712.

Murat Arikan, Reinhold Preiner, Claus Scheiblauer, Stefan Jeschke, and Michael Wimmer. 2014. Large-scale point-cloud visualization through localized textured surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 20, 9 (2014), 1280–1292.

Murat Arikan, Reinhold Preiner, and Michael Wimmer. 2015. Multi-depth-map raytracing for efficient large-scene reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 22, 2 (2015), 1127–1137.

Tim Bailey and Hugh Durrant-Whyte. 2006. Simultaneous localization and mapping (SLAM): Part II. *IEEE robotics & automation magazine* 13, 3 (2006), 108–117.

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5855–5864.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5470–5479.

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2023. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 19640–19648.

Kenneth E. Batcher. 1968. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. 307–314.

Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. 2015. Robust visual inertial odometry using a direct EKF-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 298–304.

Sebastien Bonopera, Peter Hedman, Jerome Esnault, Siddhant Prakash, Simon Rodriguez, Theo Thonat, Mehdi Benadel, Gaurav Chaurasia, Julien Philip, and George Drettakis. 2020. sibr: A System for Image Based Rendering. `https://sibr.gitlabpages.inria.fr/`

Mario Botsch, Alexander Hornung, Matthias Zwicker, and Leif Kobbelt. 2005. High-quality surface splatting on today's GPUs. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* IEEE, 17–141.

Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. 2001. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 425–432.

Giang Bui, Truc Le, Brittany Morago, and Ye Duan. 2018. Point-Based Rendering Enhancement via Deep Learning. *The Visual Computer* 34, 6 (2018), 829–841.

Thorsten M. Buzug. 2008. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*. Springer. https://books.google.de/books?id=HBfvngEACAAJ

Carlos Campos, Richard Elvira, Juan J. Gómez, José M. M. Montiel, and Juan D. Tardós. 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *IEEE Transactions on Robotics* 37, 6 (2021), 1874–1890.

Chenjie Cao, Xinlin Ren, and Yanwei Fu. 2022. MVSFormer: Multi-View Stereo by Learning Robust Image Features and Temperature-based Depth. *Transactions on Machine Learning Research* (2022), 1–13.

Chenjie Cao, Xinlin Ren, and Yanwei Fu. 2024. MVSFormer++: Revealing the Devil in Transformer's Details for Multi-View Stereo. In *International Conference on Learning Representations (ICLR)*.

Rodrigo Ortiz Cayon, Abdelaziz Djelouah, and George Drettakis. 2015. A bayesian approach for selective image-based rendering using superpixels. In *2015 International Conference on 3D Vision*. IEEE, 469–477.

Jen-Hao Rick Chang, Wei-Yu Chen, Anurag Ranjan, Kwang Moo Yi, and Oncel Tuzel. 2023. Pointersect: Neural Rendering with Cloud-Ray Intersection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8359–8369.

Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. 2013. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)* 32, 3 (2013), 1–12.

Gaurav Chaurasia, Olga Sorkine, and George Drettakis. 2011. Silhouette-Aware Warping for Image-Based Rendering. *Computer Graphics Forum* 30, 4 (2011), 1223–1232.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision (ECCV)*. Springer International Publishing, Cham, 333–350.

Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. 2021. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14124–14133.

Guikun Chen and Wenguan Wang. 2024. A survey on 3d gaussian splatting. *arXiv preprint arXiv:2401.03890* (2024).

Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019. Learning to Predict 3D Objects With an Interpolation-Based Differentiable Renderer. *Advances in Neural Information Processing Systems* 32 (2019), 9609–9619.

Yu Chen and Gim Hee Lee. 2024. DOGS: Distributed-Oriented Gaussian Splatting for Large-Scale 3D Reconstruction Via Gaussian Consensus. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16569–16578.

Shuo Cheng, Zexiang Xu, Shilin Zhu, Zhuwen Li, Li Erran Li, Ravi Ramamoorthi, and Hao Su. 2020. Deep stereo using adaptive thin volume representation with uncertainty awareness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2524–2534.

Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. 2021. Stereo radiance fields (srf): Learning view synthesis for sparse views of novel scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 7911–7920.

Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. 2016. A Large Dataset of Object Scans. *arXiv:1602.02481* (2016).

Ronald Clark. 2022. Volumetric bundle adjustment for online photorealistic scene capture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6124–6132.

Robert T Collins. 1996. A Space-Sweep Approach to True Multi-Image Matching. In *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Ieee, 358–363.

Christine A Curcio, Kenneth R Sloan, Robert E Kalina, and Anita E Hendrickson. 1990. Human Photoreceptor Topography. *Journal of Comparative Neurology* 292, 4 (1990), 497–523.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2432–2443.

Peng Dai, Yinda Zhang, Zhuwen Li, Shuaicheng Liu, and Bing Zeng. 2020. Neural Point Cloud Rendering via Multi-Plane Projection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 7830–7839.

Anurag Dalal, Daniel Hagen, Kjell G. Robbersmyr, and Kristian Muri Knausgård. 2024. Gaussian Splatting: 3D Reconstruction and Novel View Synthesis, a Review. *arXiv preprint arXiv:2405.03417* (2024).

Paul Debevec, Camillo Taylor, and Jitendra Malik. 1996. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 11–20.

Paul Debevec, Yizhou Yu, and George Boshokov. 1998. Efficient view-dependent IBR with projective texture-mapping. In *EG Rendering Workshop*, Vol. 4.

Nianchen Deng, Zhenyi He, Jiannan Ye, Budmonde Duinkharjav, Praneeth Chakravarthula, Xubo Yang, and Qi Sun. 2022. FoV-NeRF: Foveated Neural Radiance Fields for Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics* (2022).

Yikang Ding, Wentao Yuan, Qingtian Zhu, Haotian Zhang, Xiangyue Liu, Yuanjiang Wang, and Xiao Liu. 2022. Transmvsnet: Global context-aware multi-view stereo network with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8585–8594.

Joris Domhof, Julian FP Kooij, and Dariu M Gavrila. 2019. An extrinsic calibration tool for radar, camera and lidar. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 8107–8113.

Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T Barron. 2024. Smerf: Streamable memory efficient radiance fields for real-time large-scene exploration. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–13.

Andrew Edelsten, Paula Jukarainen, and Anjul Patney. 2019. Truly next-gen: Adding deep learning to games and graphics. In *NVIDIA Sponsored Sessions (Game Developers Conference)*.

Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson De Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. 2008. Floating textures. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 409–418.

Ben Fei, Jingyi Xu, Rui Zhang, Qingyuan Zhou, Weidong Yang, and Ying He. 2024. 3D Gaussian as a New Vision Era: A Survey. *arXiv preprint arXiv:2402.07181* (2024).

Laura Fink, Linus Franke, Joachim Keinert, and Marc Stamminger. 2024. Refinement of Monocular Depth Maps via Multi-View Differentiable Rendering. *arXiv preprint (arXiv:2410.03861)* (2024).

Laura Fink, Sing Chun Lee, Jie Ying Wu, Xingtong Liu, Tianyu Song, Yordanka Velikova, Marc Stamminger, Nassir Navab, and Mathias Unberath. 2019. Lumipath–towards real-time physically-based rendering on embedded devices. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part V 22*. Springer, 673–681.

Laura Fink, Darius Rückert, Linus Franke, Joachim Keinert, and Marc Stamminger. 2023a. LiveNVS: Neural View Synthesis on Live RGB-D Streams. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 37, 11 pages.

Laura Fink, Svenja Strobel, Linus Franke, and Marc Stamminger. 2023b. Efficient Rendering for Light Field Displays using Tailored Projective Mappings. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 6, 1, Article 3 (May 2023), 17 pages.

John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. 2016. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5515–5524.

Linus Franke, Laura Fink, Jana Martschinke, Kai Selgrad, and Marc Stamminger. 2021. Time-Warped Foveated Rendering for Virtual Reality Headsets. *Computer Graphics Forum* 40, 1 (2021), 110–123.

Linus Franke, Laura Fink, and Marc Stamminger. 2025. VR-Splatting: Foveated Radiance Field Rendering via 3D Gaussian Splatting and Neural Points. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 8, 1, Article 18 (May 2025), 21 pages. https://doi.org/10.1145/3728302

Linus Franke, Nikolai Hofmann, Marc Stamminger, and Kai Selgrad. 2018. Multi-layer Depth of Field Rendering with Tiled Splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1, Article 6 (July 2018), 17 pages.

Linus Franke, Darius Rückert, Laura Fink, Matthias Innmann, and Marc Stamminger. 2023. VET: Visual Error Tomography for Point Cloud Completion and High-Quality Neural Rendering. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 38, 12 pages. https://doi.org/10.1145/3610548.3618212

Linus Franke, Darius Rückert, Laura Fink, and Marc Stamminger. 2024. TRIPS: Trilinear Point Splatting for Real-Time Radiance Field Rendering. *Computer Graphics Forum* 43, 2 (2024), e15012.

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5491–5500.

Sebastian Friston, Tobias Ritschel, and Anthony Steed. 2019. Perceptual rasterization for head-mounted display image synthesis. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. 2010. Towards internet-scale multi-view stereo. In *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 1434–1441.

Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T. Freeman. 2018. Unsupervised Training for 3D Morphable Model Regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8377–8386.

Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. 2007. Multi-view stereo for community photo collections. In *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 1–8.

Dan B. Goldman, Brian Curless, Aaron Hertzmann, and Steven M. Seitz. 2009. Shape and spatially-varying brdfs from photometric stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 6 (2009), 1060–1071.

E. Bruce Goldstein and James Brockmole. 2016. *Sensation and Perception*. Cengage Learning.

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 43–54.

Jeffrey P. Grossman and William J. Dally. 1998. Point Sample Rendering. In *Eurographics Workshop on Rendering Techniques*. Springer, 181–192.

Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. 2020. Cascade cost volume for high-resolution multi-view stereo and stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2495–2504.

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D Graphics. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 10 pages.

Christian Günther, Thomas Kanzok, Lars Linsen, and Paul Rosenthal. 2013. A GPGPU-based pipeline for accelerated rendering of point clouds. *Journal of WSCG* 21, 2 (2013), 153–161.

Jie Guo, Xihao Fu, Liqiang Lin, Hengjun Ma, Yanwen Guo, Shiqiu Liu, and Ling-Qi Yan. 2021. ExtraNet: real-time extrapolated rendering for low-latency temporal supersampling. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

Florian Hahlbohm, Linus Franke, Moritz Kappel, Susana Castillo, Martin Eisemann, Marc Stamminger, and Marcus Magnor. 2025a. INPC: Implicit Neural Point Clouds for Radiance Field Rendering. In *2025 International Conference on 3D Vision (3DV)*.

Florian Hahlbohm, Fabian Friederichs, Tim Weyrich, Linus Franke, Moritz Kappel, Susana Castillo, Marc Stamminger, Martin Eisemann, and Marcus Magnor. 2025b. Efficient Perspective-Correct 3D Gaussian Splatting Using Hybrid Transparency. *Computer Graphics Forum* 44, 2 (2025).

Yuxuan Han, Ruicheng Wang, and Jiaolong Yang. 2022. Single-View View Synthesis in the Wild with Learned Adaptive Multiplane Images. In *ACM SIGGRAPH 2022 Conference Proceedings* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 14, 8 pages.

Mathias Harrer, Linus Franke, Laura Fink, Marc Stamminger, and Tim Weyrich. 2023. Inovis: Instant Novel-View Synthesis. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 14, 12 pages. https://doi.org/10.1145/3610548.3618216

Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. 2013. Rotation averaging. *International Journal of Computer Vision* 103 (2013), 267–305.

Richard Hartley and Andrew Zisserman. 2003. *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Mariam Hassan, Florent Forest, Olga Fink, and Malcolm Mielle. 2025. ThermoNeRF: A multimodal Neural Radiance Field for joint RGB-thermal novel view synthesis of building facades. *Advanced Engineering Informatics* 65 (2025), 103345.

Paul S. Heckbert. 1989. *Fundamentals of Texture Mapping and Image Warping*. Technical Report. University of California at Berkeley, USA.

Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. 2018. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)* 37, 6 (2018), 1–15.

Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. 2016. Scalable inside-out image-based rendering. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–11.

Janne Heikkila and Olli Silvén. 1997. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE, 1106–1112.

Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. 2019. Escaping Plato's Cave: 3D Shape From Adversarial Rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9984–9993.

Carlos Hernandez, George Vogiatzis, and Roberto Cipolla. 2008. Multiview photometric stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 3 (2008), 548–554.

David Hoffman, Zoe Meraz, and Eric Turner. 2018. Limits of peripheral acuity and implications for VR system design. *Journal of the Society for Information Display* 26, 8 (2018), 483–495.

Yi-Tian Hong and Han-Pang Huang. 2023. A Comparison of Outdoor 3D Reconstruction between Visual SLAM and LiDAR SLAM. In *2023 International Automatic Control Conference (CACS)*. IEEE, 1–6.

Alexander Hornung and Leif Kobbelt. 2009. Interactive Pixel-Accurate Free Viewpoint Rendering from Images with Silhouette Aware Sampling. *Computer Graphics Forum* 28, 8 (2009), 2090–2103.

HTC. 2019. HTC Vive Pro Eye Specifications. `https://www.vive.com/de/product/vive-pro-eye/`

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024b. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. In *SIGGRAPH 2024 Conference Papers (SIGGRAPH '24)*. Association for Computing Machinery, Article 32, 11 pages.

Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. 2024a. SC-GS: Sparse-Controlled Gaussian Splatting for Editable Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4220–4230.

Susmija Jabbireddy, Xuetong Sun, Xiaoxu Meng, and Amitabh Varshney. 2022. Foveated rendering: Motivation, taxonomy, and research directions. *arXiv preprint arXiv:2205.04529* (2022).

Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020. SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1251–1261.

Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, and Chenfanfu Jiang. 2024. VR-GS: A Physical Dynamics-Aware Interactive Gaussian Splatting System in Virtual Reality. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 78, 1 pages.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*. Springer, Springer International Publishing, Cham, 694–711.

Eagle S. Jones and Stefano Soatto. 2011. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research* 30, 4 (2011), 407–430.

Anton S. Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

Armin Kappeler, Seunghwan Yoo, Qiqin Dai, and Aggelos K Katsaggelos. 2016. Video super-resolution with convolutional neural networks. *IEEE transactions on computational imaging* 2, 2 (2016), 109–122.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. 2020. Differentiable Rendering: A Survey. *arXiv preprint arXiv:2006.12057* (2020).

Hiroharu Kato and Tatsuya Harada. 2019. Learning View Priors for Single-View 3D Reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9778–9787.

Benjamin Keinert, Matthias Innmann, Michael Sänger, and Marc Stamminger. 2015. Spherical fibonacci mapping. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–7.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics (TOG)* 42, 4 (July 2023). `https://doi.org/10.1145/3592433`

Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. 2024. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–15.

Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 2024. 3D Gaussian Splatting as Markov Chain Monte Carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jonghyun Kim, Youngmo Jeong, Michael Stengel, Kaan Akşit, Rachel Albert, Ben Boudaoud, Trey Greer, Joohwan Kim, Ward Lopes, Zander Majercik, Peter Shirley, Josef Spjut, Morgan McGuire, and David Luebke. 2019. Foveated AR: dynamically-foveated augmented reality display. *ACM Transactions on Graphics* 38, 4, Article 99 (July 2019), 15 pages.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. *ACM Transactions on Graphics* 36, 4 (2017).

Leif Kobbelt and Mario Botsch. 2004. A Survey of Point-Based Techniques in Computer Graphics. *Computers & Graphics* 28, 6 (2004), 801–814.

Georgios Kopanas and George Drettakis. 2023. Improving NeRF Quality by Progressive Camera Placement for Free-Viewpoint Navigation. In *Vision, Modeling, and Visualization*. The Eurographics Association.

Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. 2022. Neural Point Catacaustics for Novel-View Synthesis of Reflections. *ACM Transactions on Graphics* 41, 6 (2022).

Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. 2021. Point-Based Neural Rendering with Per-View Optimization. *Computer Graphics Forum* 40, 4 (2021), 29–43.

Matias Koskela, Timo Viitanen, Pekka Jääskeläinen, and Jarmo Takala. 2016. Foveated path tracing: a literature review and a performance gain analysis. In *Advances in Visual Computing: 12th International Symposium, ISVC 2016, Las Vegas, NV, USA, December 12-14, 2016, Proceedings, Part I 12*. Springer, 723–732.

Oliver Kreylos. 2018. The Display Resolution of Head-mounted Displays. `https://web.archive.org/web/20240723174740/http://doc-ok.org/?p=1677`.

Oliver Kreylos. 2019. Field of View and Resolution of the PlayStation VR Headset. `https://web.archive.org/web/20240614024653/https://doc-ok.org/?p=1882`.

Christoph Lassner and Michael Zollhofer. 2021. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1440–1449.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4681–4690.

Marc Levoy and Pat Hanrahan. 1996. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, New York, NY, USA, 31–42.

Marc Levoy and Turner Whitted. 1985. *The Use of Points as a Display Primitive.* Citeseer.

Noah Lewis, Darius Rückert, Marc Stamminger, and Linus Franke. 2025. NePO: Neural Point Octrees for Large-scale Novel View Synthesis. *Preprint, currently submitted* (2025).

Ke Li, Tim Rolff, Susanne Schmidt, Reinhard Bacher, Simone Frintrop, Wim Leemans, and Frank Steinicke. 2022. Immersive neural graphics primitives. *arXiv preprint arXiv:2211.13494* (2022).

Li Li, Khalid N. Ismail, Hubert .P. H. Shum, and Toby P. Breckon. 2021. DurLAR: A High-fidelity 128-channel LiDAR Dataset with Panoramic Ambient and Reflectivity Imagery for Multi-modal Autonomous Driving Applications. In *Proceedings of the International Conference on 3D Vision*. IEEE.

You Li and Javier Ibanez-Guzman. 2020. Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems. *IEEE Signal Processing Magazine* 37, 4 (2020), 50–61.

Yixuan Li, Lihan Jiang, Linning Xu, Yuanbo Xiangli, Zhenzhi Wang, Dahua Lin, and Bo Dai. 2023. Matrixcity: A large-scale city dataset for city-scale neural rendering and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3205–3215.

Jinli Liao, Yikang Ding, Yoli Shavit, Dihe Huang, Shihao Ren, Jia Guo, Wensen Feng, and Kai Zhang. 2022a. Wt-mvsnet: window-based transformers for multi-view stereo. *Advances in Neural Information Processing Systems* 35 (2022), 8564–8576.

Yiyi Liao, Jun Xie, and Andreas Geiger. 2022b. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 3 (2022), 3292–3310.

Alexander Lier, Linus Franke, Marc Marc Stamminger, and Kai Selgrad. 2016. A Case Study in Implementation-Space Exploration. In *Proceedings of the 9th European Lisp Symposium on European Lisp Symposium* (Kraków, Poland) *(ELS2016)*. European Lisp Scientific Activities Association, Article 10, 8 pages.

Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, and Wenming Yang. 2024a. VastGaussian: Vast 3D Gaussians for Large Scene Reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yvette Y. Lin, Xin-Yi Pan, Sara Fridovich-Keil, and Gordon Wetzstein. 2024b. ThermalNeRF: Thermal Radiance Fields. In *2024 IEEE International Conference on Computational Photography (ICCP)*. IEEE, 1–12.

Ce Liu and Deqing Sun. 2013. On Bayesian adaptive video super resolution. *IEEE transactions on pattern analysis and machine intelligence* 36, 2 (2013), 346–360.

Hongying Liu, Zhubo Ruan, Peng Zhao, Chao Dong, Fanhua Shang, Yuanyuan Liu, Linlin Yang, and Radu Timofte. 2022. Video super-resolution based on deep learning: a comprehensive survey. *Artificial Intelligence Review* (2022), 1–55.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. 2019a. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7708–7717.

Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. 2019b. Learning to Infer Implicit Surfaces without 3D Supervision. *Advances in Neural Information Processing Systems* 32 (2019), 8295–8306.

Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. 2020. DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019–2028.

Tianqi Liu, Xinyi Ye, Weiyue Zhao, Zhiyu Pan, Min Shi, and Zhiguo Cao. 2023. When Epipolar Constraint Meets Non-local Operators in Multi-View Stereo. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 18088–18097.

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural Volumes: Learning Dynamic Renderable Volumes From Images. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Matthew M. Loper and Michael J. Black. 2014. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*. Springer, 154–169.

David G. Lowe. 1999. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE International Conference on Computer Vision*, Vol. 2. IEEE, 1150–1157.

Chongshan Lu, Fukun Yin, Xin Chen, Wen Liu, Tao Chen, Gang Yu, and Jiayuan Fan. 2023. A Large-Scale Outdoor Multi-Modal Dataset and Benchmark for Novel View Synthesis and Implicit Scene Reconstruction. (October 2023), 7557–7567.

Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2024. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis. In *2024 International Conference on 3D Vision (3DV)*. 800–809.

Jiahao Ma, Miaomiao Liu, David Ahmedt-Aristizabal, and Chuong Nguyen. 2024. HashPoint: Accelerated Point Searching and Sampling for Neural Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4462–4472.

Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. 2024. Taming 3dgs: High-quality radiance fields with limited resources. In *SIGGRAPH Asia 2024 Conference Papers*. 1–11.

Rafał K. Mantiuk, Maliha Ashraf, and Alexandre Chapiro. 2022. stelaCSF: A unified model of contrast sensitivity as the function of spatio-temporal frequency, eccentricity, luminance and area. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Rafał K. Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. 2021. Fovvideovdp: A visible difference predictor for wide field-of-view video. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–19.

Ricardo Marroquim, Martin Kraus, and Paulo Roma Cavalcanti. 2007. Efficient Point-Based Rendering Using Image Reconstruction. In *PBG@ Eurographics*. 101–108.

Paul F. McManamon. 2019. *LiDAR Technologies and Systems*. SPIE.

Xiaoxu Meng, Ruofei Du, Joseph F JaJa, and Amitabh Varshney. 2020b. 3D-kernel foveated rendering for light fields. *IEEE Transactions on Visualization and Computer Graphics* 27, 8 (2020), 3350–3360.

Xiaoxu Meng, Ruofei Du, and Amitabh Varshney. 2020a. Eye-dominance-guided foveated rendering. *IEEE transactions on visualization and computer graphics* 26, 5 (2020), 1972–1980.

Xiaoxu Meng, Ruofei Du, Matthias Zwicker, and Amitabh Varshney. 2018. Kernel Foveated Rendering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 5.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4460–4470.

Moustafa Meshry, Dan B. Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. 2019. Neural rerendering in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6878–6887.

Lukas Meyer, Floris Erich, Yusuke Yoshiyasu, Marc Stamminger, Noriaki Ando, and Yukiyasu Domae. 2024. Pegasus: Physically enhanced gaussian splatting simulation system for 6dof object pose dataset generation. (2024), 10710–10715.

Zhenxing Mi and Dan Xu. 2023. Switch-NeRF: Learning Scene Decomposition with Mixture of Experts for Large-scale Neural Radiance Fields. In *International Conference on Learning Representations (ICLR)*.

Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (ToG)* 38, 4 (2019), 1–14.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Ravi Barron, Jonathan T.and Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision*. 405–421.

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

Thomas Müller, Alex Evans, Christoph Schied, Marco Foco, András Bódis-Szomorú, Isaac Deutsch, Michael Shelley, and Alexander Keller. 2022. Instant neural radiance fields. In *ACM SIGGRAPH 2022 Real-Time Live!* 1–2.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)* 41, 4, Article 102 (jul 2022), 15 pages.

Giljoo Nam, Joo Ho Lee, Diego Gutierrez, and Min H Kim. 2018. Practical svbrdf acquisition of 3d objects with unstructured flash photography. *ACM Transactions on Graphics (ToG)* 37, 6 (2018), 1–12.

Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. 2021. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *CGF* (2021).

Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. 2018. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 7902–7912.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3504–3515.

OpenCV 2025. *The OpenCV Reference Manual*. OpenCV.

Julian Ost, Issam Laradji, Alejandro Newell, Yuval Bahat, and Felix Heide. 2022. Neural Point Light Fields. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mert Özer, Maximilian Weiherer, Martin Hundhausen, and Bernhard Egger. 2025. Exploring Multimodal Neural Scene Representations With Applications on Thermal Imaging. In *Computer Vision – ECCV 2024 Workshops*. Springer Nature Switzerland, Cham, 82–98.

Linfei Pan, Daniel Barath, Marc Pollefeys, and Johannes Lutz Schönberger. 2024. Global Structure-from-Motion Revisited. In *European Conference on Computer Vision (ECCV)*. Springer International Publishing, Cham.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 165–174.

Janusch Patas. 2023. Gaussian Splatting Cuda. `https://github.com/MrNeRF/gaussian-splatting-cuda`.

Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards Foveated Rendering for Gaze-Tracked Virtual Reality. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 179.

Eric Penner and Li Zhang. 2017. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–11.

Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. 2000. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 335–342.

Julien Philip and Valentin Deschaintre. 2023. Floaters No More: Radiance Field Gradient Scaling for Improved Near-Camera Training. In *Eurographics Symposium on Rendering*, Tobias Ritschel and Andrea Weidlich (Eds.). The Eurographics Association.

Ruggero Pintus, Enrico Gobbetti, and Marco Agus. 2011. Real-time rendering of massive unstructured raw point clouds using screen-space operators. In *Proceedings of the 12th International conference on Virtual Reality, Archaeology and Cultural Heritage*. 105–112.

Francesco Pittaluga, Sanjeev J. Koppal, Sing Bing Kang, and Sudipta N. Sinha. 2019. Revealing Scenes by Inverting Structure From Motion Reconstructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Thomas Porter and Tom Duff. 1984. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 253–259.

Florent Poux, Christian Mattes, Zain Selman, and Leif Kobbelt. 2022. Automatic region-growing system for the segmentation of large point clouds. *Automation in Construction* 138 (2022), 104250.

Reinhold Preiner, Stefan Jeschke, and Michael Wimmer. 2012. Auto Splats: Dynamic Point Cloud Visualization on the GPU.. In *EGPGV@ Eurographics*. 139–148.

Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. 2024. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–17.

Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. 2022. NPBG++: Accelerating Neural Point-Based Graphics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 15948–15958.

Christian Reiser, Stephan Garbin, Pratul P. Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan Barron, Peter Hedman, and Andreas Geiger. 2024. Binary Opacity Grids: Capturing Fine Geometric Detail for Mesh-Based View Synthesis. *ACM Transactions on Graphics (TOG)* 43, 4, Article 149 (July 2024), 14 pages.

Christian Reiser, Rick Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. 2023. MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *ACM Transactions on Graphics (TOG)* 42, 4, Article 89 (July 2023), 12 pages.

Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. 2025. Octree-GS: Towards Consistent Real-time Rendering with LOD-Structured 3D Gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025), 1–15.

Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. 2015. A Versatile Scene Model With Differentiable Visibility Applied to Generative Pose Estimation. In *Proceedings of the IEEE International Conference on Computer Vision*. 765–773.

Gernot Riegler and Vladlen Koltun. 2020. Free view synthesis. In *European Conference on Computer Vision (ECCV)*. Springer, Springer International Publishing, Cham, 623–640.

Gernot Riegler and Vladlen Koltun. 2021. Stable view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12216–12225.

Tim Rolff, Ke Li, Julia Hertel, Susanne Schmidt, Simone Frintrop, and Frank Steinicke. 2023a. Interactive VRS-NeRF: Lightning fast Neural Radiance Field Rendering for Virtual Reality. In *Proceedings of the 2023 ACM Symposium on Spatial User Interaction*. 1–3.

Tim Rolff, Susanne Schmidt, Ke Li, Frank Steinicke, and Simone Frintrop. 2023b. VRS-NeRF: Accelerating Neural Radiance Field Rendering with Variable Rate Shading. In *2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 243–252.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, Cham, 234–241.

Paul Rosenthal and Lars Linsen. 2008. Image-space point cloud rendering. In *Proceedings of Computer Graphics International*. Citeseer, 136–143.

Darius Rückert, Linus Franke, and Marc Stamminger. 2022. ADOP: Approximate Differentiable One-Pixel Point Rendering. *ACM Transactions on Graphics (TOG)* 41, 4, Article 99 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530122

Darius Rückert and Marc Stamminger. 2021. Snake-SLAM: Efficient global visual inertial SLAM using decoupled nonlinear optimization. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 219–228.

Darius Rückert, Yuanhao Wang, Rui Li, Ramzi Idoughi, and Wolfgang Heidrich. 2022. NeAT: Neural Adaptive Tomography. *ACM Transactions on Graphics (TOG)* 41, 4, Article 55 (jul 2022), 13 pages.

Joaquim Salvi, Xavier Armangué, and Joan Batlle. 2002. A comparative review of camera calibrating methods with accuracy evaluation. *Pattern recognition* 35, 7 (2002), 1617–1635.

Daniel Scharstein and Richard Szeliski. 2003. High-Accuracy Stereo Depth Maps Using Structured Light. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, Vol. 1. IEEE, I–I.

Johannes Lutz Schönberger. 2024. *COLMAP project page.* `https://colmap.github.io/index.html`

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4104–4113.

Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*. Springer International Publishing, Cham, 501–518.

Markus Schütz. 2016. Potree: Rendering large point clouds in web browsers. *Technische Universität Wien* (2016).

Markus Schütz, Bernhard Kerbl, and Michael Wimmer. 2022. Software rasterization of 2 billion points in real time. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 3 (2022), 1–17.

Markus Schütz, Stefan Ohrhallinger, and Michael Wimmer. 2020. Fast Out-of-Core Octree Generation for Massive Point Clouds. *Computer Graphics Forum* 39, 7 (2020), 155–167.

Markus Schütz, Bernhard Kerbl, and Michael Wimmer. 2021. Rendering Point Clouds with Compute Shaders and Vertex Order Optimization. *Computer Graphics Forum* 40, 4 (2021), 115–126.

Kai Selgrad, Carsten Dachsbacher, Quirin Meyer, and Marc Stamminger. 2015. Filtering multi-layer shadow maps for accurate soft shadows. *Computer Graphics Forum* 34, 1 (2015), 205–215.

Kai Selgrad, Linus Franke, and Marc Stamminger. 2016. Tiled depth of field splatting. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Posters* (Lisbon, Portugal) *(EG '16)*. Eurographics Association, Goslar, DEU, 39–40.

Xuehuai Shi, Lili Wang, Xinda Liu, Jian Wu, and Zhiwen Shao. 2024. Scene-aware Foveated Neural Radiance Fields. *IEEE Transactions on Visualization and Computer Graphics* (2024), 1–14.

Harry Shum and Sing Bing Kang. 2000. Review of image-based rendering techniques. In *Visual Communications and Image Processing 2000*, Vol. 4067. SPIE, 2–13.

Heung-Yeung Shum, Shing-Chow Chan, and Sing Bing Kang. 2008. *Image-based rendering*. Springer Science & Business Media.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems* 33 (2020).

Vincent Sitzmann, Ana Serrano, Amy Pavel, Maneesh Agrawala, Diego Gutierrez, Belen Masia, and Gordon Wetzstein. 2018. Saliency in VR: How do people explore virtual environments? *IEEE transactions on visualization and computer graphics* 24, 4 (2018), 1633–1642.

Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. 2019. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2437–2446.

Randall Smith, Matthew Self, and Peter Cheeseman. 1990. Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*. Springer, 167–193.

Noah Snavely, Steven M. Seitz, and Richard Szeliski. 2006. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics (TOG)* 25, 3 (July 2006), 835–846.

Pratul P. Srinivasan, Richard Tucker, Ravi Barron, Jonathan T.and Ramamoorthi, Ren Ng, and Noah Snavely. 2019. Pushing the boundaries of view extrapolation with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 175–184.

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive Image-Space Sampling for Gaze-Contingent Real-Time Rendering. , 129–139 pages.

Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison. 2021. iMAP: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6229–6238.

Qi Sun, Fu-Chung Huang, Joohwan Kim, Li-Yi Wei, David Luebke, and Arie Kaufman. 2017. Perceptually-guided foveation for light field displays. *ACM Transactions on Graphics (TOG)* 36, 6, Article 192 (2017), 13 pages.

Nicholas T. Swafford, José A Iglesias-Guitian, Charalampos Koniaris, Bochang Moon, Darren Cosker, and Kenny Mitchell. 2016. User, metric, and computational evaluation of foveated rendering methods. In *Proceedings of the ACM Symposium on Applied Perception*. 7–14.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. 2022. Block-NeRF: Scalable Large Scene Neural View Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8248–8258.

Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, and Ren Barron, Jonathan T.and Ng. 2021. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2846–2855.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 632, 11 pages.

Pingbo Tang, Daniel Huber, and Burcu Akinci. 2007. A comparative analysis of depth-discontinuity and mixed-pixel detection algorithms. In *Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM 2007)*. IEEE, 29–38.

Xin Tao, Hongyun Gao, Renjie Liao, Jue Wang, and Jiaya Jia. 2017. Detail-revealing deep video super-resolution. In *Proceedings of the IEEE International Conference on Computer Vision*. 4472–4480.

Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, Rohit Pandey, Sean Fanello, Gordon Wetzstein, Jun-Yan Zhu, Christian Theobalt, Maneesh Agrawala, Eli Shechtman, Dan B. Goldman, and Michael Zollhöfer. 2020. State of the Art on Neural Rendering. *Computer Graphics Forum* 39, 2, 701–727.

Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Niessner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhoefer, and Vladislav Golyanik. 2022. Advances in Neural Rendering. *Computer Graphics Forum* 41, 2, 703–735.

Justus Thies, Michael Zollhöfer, and Matthias Nießner. 2019. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Justus Thies, Michael Zollhöfer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. 2020. Image-guided Neural Object Rendering. In *International Conference on Learning Representations*.

Manu Mathew Thomas, Gabor Liktor, Christoph Peters, Sungye Kim, Karthik Vaidyanathan, and Angus G Forbes. 2022. Temporally Stable Real-Time Joint Neural Denoising and Supersampling. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 3 (2022), 1–22.

Richard Tucker and Noah Snavely. 2020. Single-view view synthesis with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 551–560.

Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. 2017. Multi-View Supervision for Single-View Reconstruction via Differentiable Ray Consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2626–2634.

Haithem Turki, Vasu Agrawal, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Deva Ramanan, Michael Zollhöfer, and Christian Richardt. 2024. HybridNeRF: Efficient Neural Rendering via Adaptive Volumetric Surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19647–19656.

Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. 2022. Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12922–12931.

Okan Tarhan Tursun, Elena Arabadzhiyska-Koleva, Marek Wernikowski, Radosław Mantiuk, Hans-Peter Seidel, Karol Myszkowski, and Piotr Didyk. 2019. Luminance-contrast-aware foveated rendering. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Andreas-Alexandros Vasilakis, Konstantinos Vardis, and Georgios Papaioannou. 2020. A survey of multifragment rendering. *Computer Graphics Forum* 39, 2 (2020), 623–642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

Sundar Vedula, Simon Baker, and Takeo Kanade. 2002. Spatio-temporal view interpolation. *Rendering Techniques* 28 (2002), 65–76.

Ulla Wandinger. 2005. Introduction to Lidar. In *Lidar: Range-Resolved Optical Remote Sensing of the Atmosphere*. Springer, 1–18.

Lili Wang, Xuehuai Shi, and Yi Liu. 2023a. Foveated rendering: A state-of-the-art survey. *Computational Visual Media* 9, 2 (2023), 195–228.

Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P. Srinivasan, Howard Zhou, Ricardo Barron, Jonathan T.and Martin-Brualla, Noah Snavely, and Thomas Funkhouser. 2021. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4690–4699.

Yuesong Wang, Zhaojie Zeng, Tao Guan, Wei Yang, Zhuo Chen, Wenkai Liu, Luoyuan Xu, and Yawei Luo. 2023b. Adaptive patch deformation for textureless-resilient multi-view stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1621–1630.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

Zijun Wang, Jian Wu, Runze Fan, Wei Ke, and Lili Wang. 2024. VPRF: Visual Perceptual Radiance Fields for Foveated Image Synthesis. *IEEE Transactions on Visualization and Computer Graphics* 30, 11 (Nov. 2024), 7183–7192.

Zizhuang Wei, Qingtian Zhu, Chen Min, Yisong Chen, and Guoping Wang. 2021. AA-RMVSNet: Adaptive Aggregation Recurrent Multi-View Stereo Network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 6187–6196.

Martin Weier, Thorsten Roth, André Hinkenjann, and Philipp Slusallek. 2018. Foveated Depth-of-Field Filtering in Head-Mounted Displays. *ACM Transactions on Applied Perception (TAP)* 15, 4 (2018), 26.

Martin Weier, Thorsten Roth, Ernst Kruijff, André Hinkenjann, Arsène Pérard-Gayot, Philipp Slusallek, and Yongmin Li. 2016. Foveated Real-Time Ray Tracing for Head-Mounted Displays. *Computer Graphics Forum* 35, 7 (2016), 289–298.

Martin Weier, Michael Stengel, Thorsten Roth, Piotr Didyk, Elmar Eisemann, Martin Eisemann, Steve Grogorick, André Hinkenjann, Ernst Kruijff, Marcus Magnor, Karol Myszkowski, and Philipp Slusallek. 2017. Perception-Driven Accelerated Rendering. *Computer Graphics Forum* 36, 2 (2017), 611–643.

Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. 2020. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7467–7477.

Vanessa Wirth, Johanna Bräunig, Danti Khouri, Florian Gutsche, Martin Vossiek, Tim Weyrich, and Marc Stamminger. 2024. Automatic Spatial Calibration of Near-Field MIMO Radar With Respect to

Optical Depth Sensors. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 8322–8329.

Oliver J. Woodman. 2007. *An introduction to inertial navigation.* Technical Report. University of Cambridge, Computer Laboratory.

Rundi Wu, Ben Mildenhall, Philipp Henzler, Keunhong Park, Ruiqi Gao, Daniel Watson, Pratul P. Srinivasan, Dor Verbin, Jonathan T. Barron, Ben Poole, and Aleksander Holynski. 2024a. ReconFusion: 3D Reconstruction with Diffusion Priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 21551–21561.

Tong Wu, Yu-Jie Yuan, Ling-Xiao Zhang, Jie Yang, Yan-Pei Cao, Ling-Qi Yan, and Lin Gao. 2024b. Recent advances in 3d gaussian splatting. *Computational Visual Media* (2024), 1–30.

Rui Xia, Yue Dong, Pieter Peers, and Xin Tong. 2016. Recovering shape and spatially-varying surface reflectance under unknown illumination. *ACM Transactions on Graphics (ToG)* 35, 6 (2016), 1–12.

Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural supersampling for real-time rendering. *ACM Transactions on Graphics (TOG)* 39, 4, Article 142 (Aug. 2020), 12 pages.

Linning Xu, Vasu Agrawal, William Laney, Tony Garcia, Aayush Bansal, Changil Kim, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Aljaž Božič, Dahua Lin, Michael Zollhöfer, and Christian Richardt. 2023. VR-NeRF: High-Fidelity Virtualized Walkable Spaces. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 43, 12 pages.

Qingshan Xu, Weihang Kong, Wenbing Tao, and Marc Pollefeys. 2022a. Multi-Scale Geometric Consistency Guided and Planar Prior Assisted Multi-View Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 4 (2022), 4945–4963.

Qingshan Xu and Wenbing Tao. 2019. Multi-Scale Geometric Consistency Guided Multi-View Stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5483–5492.

Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. 2022b. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5438–5448.

Jianfeng Yan, Zizhuang Wei, Hongwei Yi, Mingyu Ding, Runze Zhang, Yisong Chen, Guoping Wang, and Yu-Wing Tai. 2020. Dense hybrid recurrent multi-view stereo net with dynamic consistency checking. In *European Conference on Computer Vision (ECCV)*. Springer, Springer International Publishing, Cham, 674–689.

Jiayu Yang, Wei Mao, Jose M Alvarez, and Miaomiao Liu. 2020c. Cost volume pyramid based depth inference for multi-view stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4877–4886.

Lei Yang, Shiqiu Liu, and Marco Salvi. 2020b. A survey of temporal antialiasing techniques. *Computer Graphics Forum* 39, 2 (2020), 607–621.

Zhenpei Yang, Yuning Chai, Dragomir Anguelov, Yin Zhou, Pei Sun, Dumitru Erhan, Sean Rafferty, and Henrik Kretzschmar. 2020a. SurfelGAN: Synthesizing Realistic Sensor Data for Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. 2024. Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20331–20341.

Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. 2018. Mvsnet: Depth Inference for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*. Springer International Publishing, Cham, 767–783.

Yao Yao, Zixin Luo, Shiwei Li, Tianwei Shen, Tian Fang, and Long Quan. 2019. Recurrent mvsnet for high-resolution multi-view stereo depth inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5525–5534.

Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. 2023. BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 46, 9 pages.

Jiannan Ye, Xiaoxu Meng, Daiyun Guo, Cheng Shang, Haotian Mao, and Xubo Yang. 2024b. Neural foveated super-resolution for real-time VR rendering. *Computer Animation and Virtual Worlds* 35, 4 (2024), e2287.

Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. 2024a. AbsGS: Recovering Fine Details in 3D Gaussian Splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia* (Melbourne VIC, Australia) *(MM '24)*. Association for Computing Machinery, New York, NY, USA, 1053–1061.

Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. 2019. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics* 38, 6, Article 230 (Nov. 2019), 14 pages.

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021a. PlenOctrees for Real-Time Rendering of Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5752–5761.

Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. 2021b. pixelNeRF: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4578–4587.

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. 2019. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE/CVF international conference on computer vision*. 4471–4480.

Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. 2024. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 19447–19456.

Zhenlong Yuan, Jiakai Cao, Zhaoqi Wang, and Zhaoxin Li. 2024. TSAR-MVS: Textureless-aware segmentation and correlative refinement guided multi-view stereo. *Pattern Recognition* 154, C (Oct. 2024), 15 pages.

Sergey Zakharov, Wadim Kehl, Arjun Bhargava, and Adrien Gaidon. 2020. Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12224–12233.

Pietro Zanuttigh, Giulio Marin, Carlo Dal Mutto, Fabio Dominio, Ludovico Minto, and Guido Maria Cortelazzo. 2016. *Time-of-Flight and Structured Light Depth Cameras: Technology and Applications* (1st ed.). Springer Publishing Company, Incorporated.

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492* (2020).

Qiang Zhang, Seung-Hwan Baek, Szymon Rusinkiewicz, and Felix Heide. 2022. Differentiable Point-Based Radiance Fields for Efficient View Synthesis. In *SIGGRAPH Asia 2022 Conference Papers* (Daegu, Republic of Korea) *(SA '22)*. Association for Computing Machinery, New York, NY, USA, Article 7, 12 pages.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yanshu Zhang, Shichong Peng, Seyed Alireza Moazenipourasil, and Ke Li. 2023. PAPR: Proximity Attention Point Rendering. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhengyou Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence* 22, 11 (2000), 1330–1334.

Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. 2018. Stereo magnification: learning view synthesis using multiplane images. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A. Efros. 2016. View synthesis by appearance flow. In *European Conference on Computer Vision (ECCV)*. Springer, Springer International Publishing, Cham, 286–301.

Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. 2018. Visual Object Networks: Image Generation with Disentangled 3D Representations. *Advances in Neural Information Processing Systems* 31 (2018), 118–129.

Qingtian Zhu, Chen Min, Zizhuang Wei, Yisong Chen, and Guoping Wang. 2021. Deep Learning for Multi-View Stereo via Plane Sweep: A Survey. *arXiv preprint arXiv:2106.15328* (2021).

C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. 2004. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 600–608.

Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. 2018. State of the Art on 3D Reconstruction with RGB-D Cameras. *Computer Graphics Forum* 37, 2 (2018), 625–652.

Yiming Zuo and Jia Deng. 2023. View Synthesis with Sculpted Neural Points. In *International Conference on Learning Representations (ICLR)*.

Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. 2001. Surface Splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 371–378.