Nearfield Viewer - Report

David Köppl

September 11, 2025

Abstract

In lighting design, simulations rely on far-field photometry, which represents a luminaire by a point source with 2D angular luminous-intensity distribution. Moreover, lighting tools focus on luminaire's illumination while neglecting their visual appearance. For luminaires with optically complex fixtures, far-field data is inaccurate. Near-field scans capture the required detail, but have large memomry requirements and are slow to render using current approaches. We present a real-time viewer that displays the luminaire appearance directly from near-field measurements rather than running a global illumination simulation. For every screen pixel, we find the best measurement point and sample the corresponding data. Through our viewer, we showcase that near-field data can be used interactively on consumer hardware by using our data packing and sampling approach. Our approach provides a practical baseline for future work on neural compression and using higher resolution scans, aiming to further improve visual quality in lighting design.

Contents

1 Introduction

Traditionally, lighting design software uses far-field representations to render a large number of luminaires efficiently. Far-field representations are acquired by goniophotometric measurements, where the target light source is measured with a light sensor at a fixed distance, rotating over multiple angles around the light. When the fixed distance is larger than the limiting photometric distance (LPD), the measurement is considered to be far-field; otherwise, it is considered near-field. Because the LPD depends on a light's dimensions, larger ones require a greater measurement distance. Moreover, far-field representations are accurate to within application tolerances, mainly for observation distances smaller than the LPD. Therefore, far-field representations provide inaccurate results for surfaces that are closer than the LPD to the light source [1].

Using near-field representations presents a promising solution. In contrast to far-field representations, they not only capture radiance near the source and can be evaluated for both near and far observers, but they also require less lab space for measurement. Moreover, the LPD can be derived from near-field data, are therefore often available before a far-field measurement is made [1, 2].

Unfortunately, two key challenges for near-field photometry remain unsolved. First, an interactive lighting simulation based on near-field measurements is still beyond the reach of current methods. Second, existing tools rarely render the luminaire's geometry from the measured luminance values directly.

In this work, we address the second problem: we investigate near-field goniophotometric data and design a viewer that displays the luminaire itself, from any viewpoint, by sampling the recorded ray images (Fig. 1) in real time. By making the raw data explorable on common hardware, the viewer provides a foundation which future research can build upon.

In this report, we discuss recent research, present our data set, and explain our method for rendering it. Moreover, we will highlight strengths and weaknesses in our approach and touch on future research possibilities.

2 Related Work

Available rendering methods for near-field data are used in domains such as automotive lighting, to precisely simulate headlights [3, 4]. Goniophotometric scans provide emission rays (origins, directions) among other properties, together with the luminaire's geometry. These rays define the source's near-field emission.

For scene rendering, forward path tracing is commonly used to compute a single camera view given a geometric/optical model of the luminaire. In contrast, photon-based methods (photon tracing/photon mapping) emit photons from the source according to the emitter's directional distribution, store surface interactions, and then estimate radiance along camera

rays by density estimation. This can also be used to populate the surrounding light field in near-field contexts [4, 5].

While photon mapping was initially slow, taking minutes to hours for a single frame, with advancements such as hardware-accelerated ray-tracing, it has become viable for interactive visualizations. Smal et al. report 21.7 ms per frame for a benchmark with two light sources and 1 M photons on an NVIDIA RTX 2080 Ti at Full HD. However, architectural scenes typically contain many luminaires with complex geometry, which increases the required photon count substantially. Therefore, we expect that interactive frame rates are unlikely without strong approximations [6].

Velázquez-Armendáriz et al. [7] use the photon mapping method in a precomputation step to build anisotropic point lights (APLs) for illumination. Moreover, they use limited-bounce ray-tracing and spherical harmonics to render the luminaire's appearance. However, their method takes multiple minutes for a single light source, making their approach only suitable for offline rendering [7].

The method presented in Zhu et. al. [8] uses neural representations to render complex luminaires. Here, three neural networks are trained, one for querying the lightfield, one for importance sampling, and one for transparency. Using path tracing, the neural representation is faster to render than the regular luminaire geometry. The neural representation requires only a few MB, but the path-traced render times reported for a single luminaire remain minutes per image. Further acceleration would be required for real-time usage [8].

3 Problem Statement

For our approach, we use near-field data of an Ambitus luminaire [9]. The dataset consists of two ray files produced by a commercial goniophotometer (TechnoTeam RIGO801 [10]) using an LMK-6-12 photometer at a capture radius of $r_s = 1.117m$. The goniophotometer places the camera on a capture sphere of radius r_s around the luminaire, sampling discrete positions parameterized by spherical angles (θ,φ). In our data set, one ray file covers the upper hemisphere and the other the lower hemisphere. Each ray file contains about 28,000 ray images. One ray image stores the per-pixel luminance measured from a specific camera position on the capture sphere. Note that for each ray image, there exists a unique camera position on the capture sphere. Our ray files consist of multiple ray images with dimensions of 1216×1021 pixels (Fig. 1).

Figure 1: Unprocessed ray image from a fixed camera position of the Ambitus luminaire. The measurement setup introduces reflections that appear as artifacts at the bottom of the image. For photos of the luminaire's geometry, see the product gallery [9]

For our real-time viewer, we need to access all ray images on the GPU. Storing all images as single-channel 16-bit floats would require $\approx 129\,GiB$ for 56,000 ray images, excluding mip levels. This presents a problem, as currently, consumer graphics cards do not provide this amount of VRAM.

Furthermore, since the ray images are captured at discrete positions, but our viewing angles can be arbitrary, we have to consider interpolation and expect angular sampling artifacts. Moreover, our viewer must be able to display the near-field data even if the user camera moves inside the capture sphere.

Therefore, our approach must be able to interpolate ray images based on viewing angle, support extrapolating data for inside-sphere views, and ideally require less than a few GB of VRAM per luminaire. Additionally, it must run at least at 30 frames per second (fps) and produce a comparable visual quality to the captured ray images. We will also convert the per-pixel luminous flux to candela per square meter and tone-map the values for display using the Reinhard operator [11].

4 Baseline Method

4.1 Overview

In order to visualize a ray file, we utilize the fact that the ray images lie on a capture sphere of radius r_s. For each pixel, we cast a frustum-aligned view ray and intersect it with the capture sphere (Fig. 2). From the hit point, we select the nearest measured camera and rotate

the view ray into that camera's local frame. We then project to image space, sample the corresponding pixel, convert it to luminance, and tone map it before display (Fig. 3).

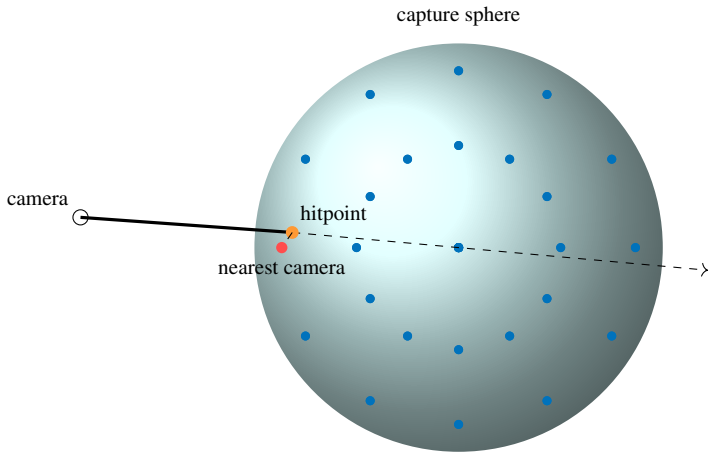


Figure 2: Finding nearest ray image for a view ray by intersecting it with the capture sphere.

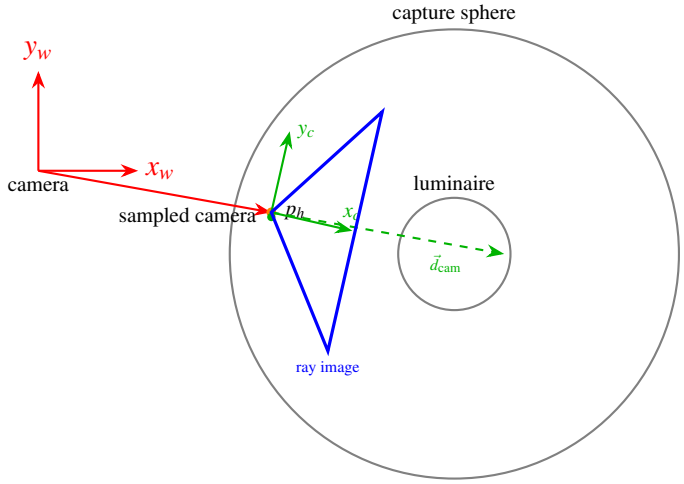


Figure 3: Red: world-space camera frame and a view ray intersecting the capture sphere at p_h. Green: nearest sampled-camera local frame (x_c, y_c) and the view ray expressed in that frame. Blue: the ray-image plane for that camera; the dashed green ray clearly intersects the plane, indicating where the ray image is sampled.

4.2 Implementation Details

First, our method intersects a view ray with the capture sphere. From the intersection points, we choose the entry point (closest intersection to the eye) as the hit point. Mapping a hit point to candidate cameras requires the ray-file index data on the GPU. We assign each camera and associated ray image an index. All hit points lie on the sphere, so we parameterize them by spherical angles ϕ, θ. To only handle positive values, we use degrees with $\phi \in [0, 360)$

and $\theta \in [0, 180)$. Then a 2D look-up-table (LUT) of size 360x180 maps all (ϕ, θ) bins on the sphere to a value we define.

Ideally, each (ϕ, θ) bin would contain exactly one camera. In practice, the capture is non-uniform, so bins may be empty or contain several cameras; therefore, we add another indirection. For each (ϕ, θ) bin, we collect camera indices from a 1° neighborhood of bins and append them to a GPU array. Then we fill our LUT with the offset into that array and the number of collected camera indices as illustrated in Fig. 4. That way, each (ϕ, θ) bin maps to a set of candidate cameras.

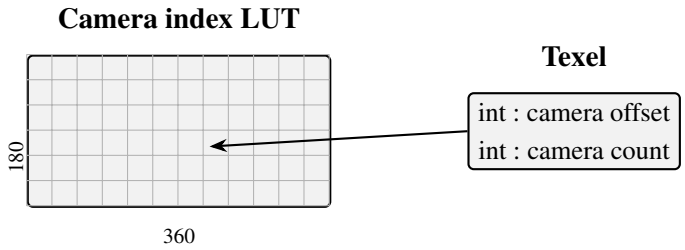


Figure 4: The camera-index LUT (180×360 texels) stores an *offset, count* pair that selects a range of camera IDs in the camera-index buffer.

Now we associate each hit point with a set of camera indices. To find the camera with the closest position to our hit point and minimize our discretization error (Fig. 5), we provide arrays that contain the ϕ_i and θ_i values of the captured cameras. We iterate over the candidate indices and select the camera minimizing a planar angular distance:

$$\Delta\phi_i(\phi) = \Big[\big(\phi_i - \phi + 180^\circ\big) \bmod 360^\circ\Big] - 180^\circ, \tag{1}$$

$$\Delta\theta_i(\theta) = \theta_i - \theta, \tag{2}$$

$$e_i(\phi, \theta) = \Delta\phi_i(\phi)^2 + \Delta\theta_i(\theta)^2, \tag{3}$$

$$\text{best}(\phi, \theta) = \underset{i \in \mathscr{C}(\phi,\theta)}{\arg\min}\, e_i(\phi, \theta). \tag{4}$$

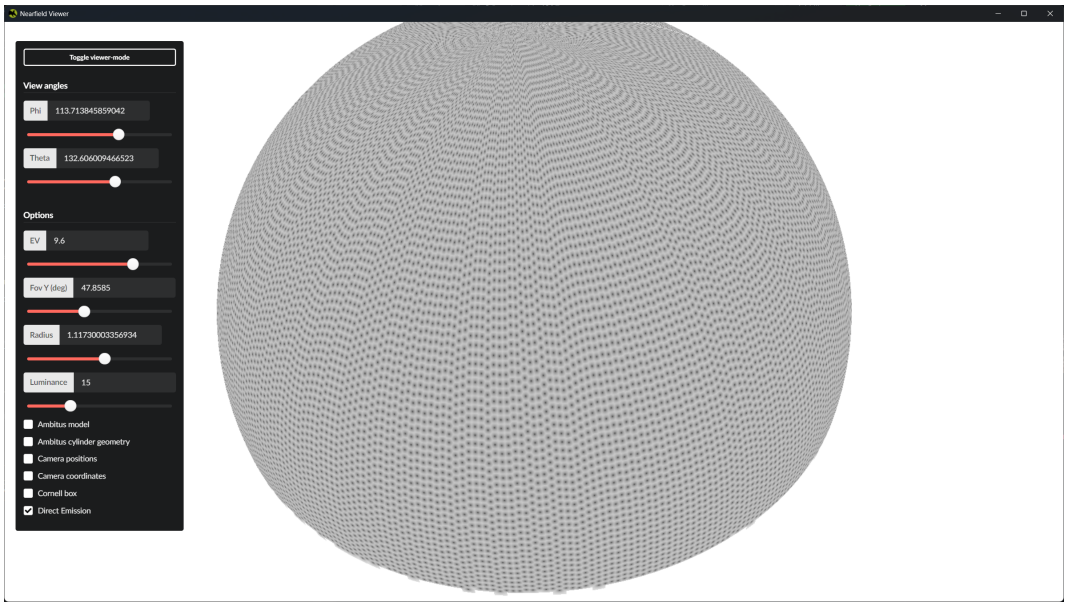


Figure 5: Angular distance from each hit point to the chosen camera. Black dots indicate exact matches. The non-uniform distribution is obvious. The camera density increases toward the top of the sphere.

Afterwards, using the camera index, we read the camera's coordinate basis, form a view matrix $V \in \mathbb{R}^{3\times3}$ and rotate the view ray d_{view} into local camera space:

$$d_{\text{cam}} = V\, d_{\text{view}}.$$

To compute the texture coordinates uv we need to transform the camera-space ray d_{cam} into homogenous coordinates x_{ndc} using a projection matrix $P \in \mathbb{R}^{4\times4}$. This matrix is constructed from a ray image's coordinate center and its pixel scaling. We use the same projection for all ray images, since the camera intrinsics stay constant. We compute the uv to sample the ray image by:

$$\tilde{x} = P \begin{bmatrix} d_{\text{cam}} \\ 1 \end{bmatrix}, \qquad uv = \frac{1}{2}\left(\begin{bmatrix} \tilde{x}_x/\tilde{x}_w \\ \tilde{x}_y/\tilde{x}_w \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right).$$

As mentioned in Sec. 3, we cannot upload ray images directly as textures on the GPU. Luckily, the ray images consist largely of black pixels, indicating a lack of captured rays there. Or put differently, ray images can be interpreted as sparse matrices. We utilize this fact by storing the raw image data on the GPU in compressed sparse row (CSR) format, which is lossless, using storage buffers instead of textures. Thereby reducing the memory footprint from $\approx 130\,GB$ down to $\approx 4.5\,GB$ for Ambitus. We concatenated the CSR data into three arrays (row, column, value) and added a small header (rowOffset, rowCount, columnOffset) for each ray image to index each CSR-compressed image. This approach has the disadvantage that the pre-processing time needs an additional 7-8 minutes on an AMD Ryzen 5600X CPU. We addressed this by implementing a naive caching system. Therefore, we only build the CSR data once and can then instantly upload it to GPU buffers in subsequent runs.

We implemented our ray image viewer in Aardvark [12] using the F# programming language. We chose Vulkan because it supports larger storage buffers on our hardware than OpenGL. Additionally, we built a wrapper around the TechnoTeam ray data C API, which is part of their ray file converter software [13], to load and process ray files in Aardvark.

5 Results

With our method explained, we can look at the results of our near-field data viewer and discuss our findings and limitations.

First, let us look at the distribution of camera positions in our ray files. For that, we load both hemispheres and render each camera position as a point. The goniophotometer samples along meridians (approximately constant ϕ), producing visible lines from pole to equator. However, samples are not aligned along parallels (constant θ), produce the jitter seen in Fig. 6. For future rendering approaches, this pattern might need to be considered to avoid certain biases.

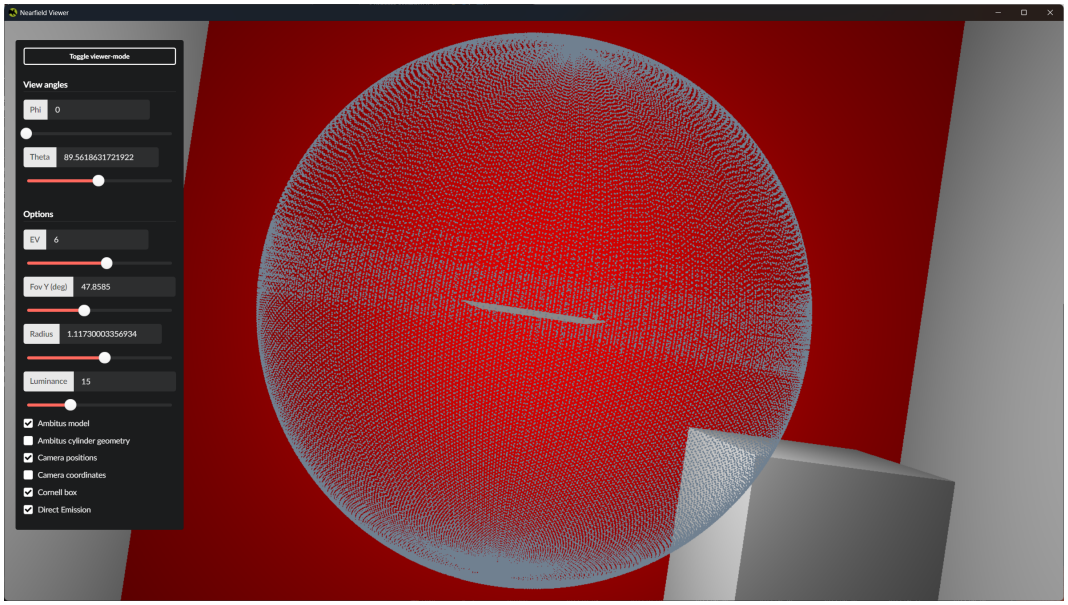
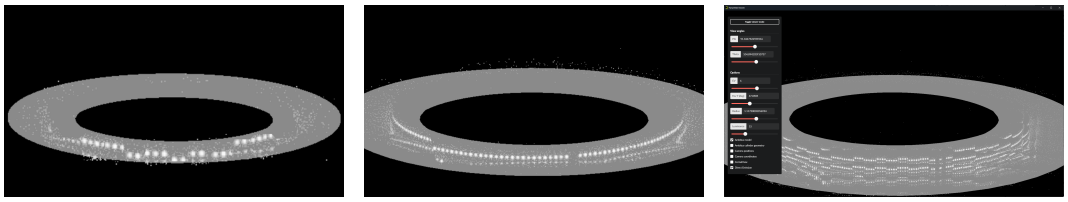


Figure 6: Camera positions of both upper and lower hemisphere ray files rendered as points showing the distribution of camera positions. Note the overlap of points around the equator. Moreover, the points do not lie on a grid and are denser at the poles.

Looking at the result of our ray image viewer in Fig. 7, we see the same ray file rendered from different camera distances. Starting with the far view (Fig. 7a), some parts of the luminaire appear as patches where no measured camera covers the required directions, reflecting the limited angular coverage per image.

In the middle view (Fig. 7b), where the user camera roughly aligns with a ray image, the visualization is more continuous. Of course, any artifact captured in the ray image, such as the noise around the rim, is also present.

When we move the camera closer than the capture sphere radius (Fig. 7c), we start to see a parallax error creating duplication of light patterns. This can be addressed by a geometry-aware approach that utilizes a known proxy geometry of the luminaire as described in Sec. 6.



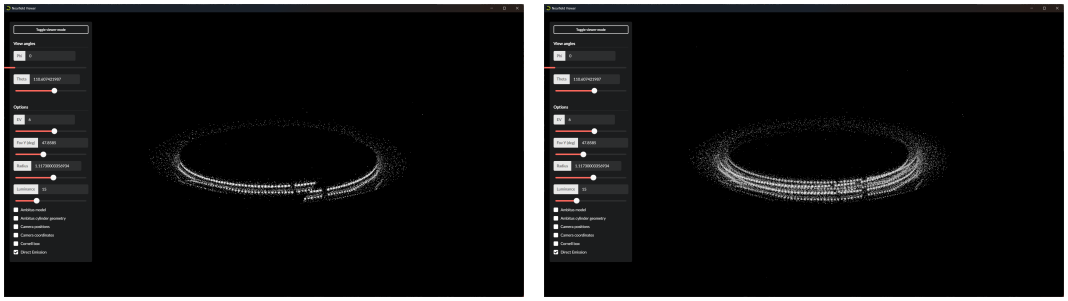
(a) Far view $r_s < d = 2.18$. (b) Middle view $r_s \approx d = 1.17$. (c) Near view $r_s > d = 0.58$.

Figure 7: Comparison of our ray image viewing method at different camera distances d with the capture sphere radius r_s. The Ambitus geometry was added as a size reference.

We also experimented with interpolating data from multiple ray images. Instead of selecting the nearest camera index, we choose the four nearest camera indices and weight them by their normalized inverse distances:

$$\tilde{w}_k = \frac{1}{\varepsilon_k}, \qquad k \in \{0,1,2,3\}, \qquad w_k = \frac{\tilde{w}_k}{\sum_{j=0}^{3} \tilde{w}_j}, \qquad \sum_{k=0}^{3} w_k = 1. \tag{5}$$

The result is shown in Fig. 8. We see that linear interpolation produces smoother continuity but increases noise due to cross-image inconsistencies and misalignment.



(a) Nearest (b) Linear

Figure 8: Comparison of choosing the nearest camera and linearly interpolating between the four nearest cameras.

Regarding performance, our ray image viewer renders at 60 fps at 1920×1080 on an RTX 3070 (8 GB) with about 28,000 ray images (CSR footprint 2.7 GB).

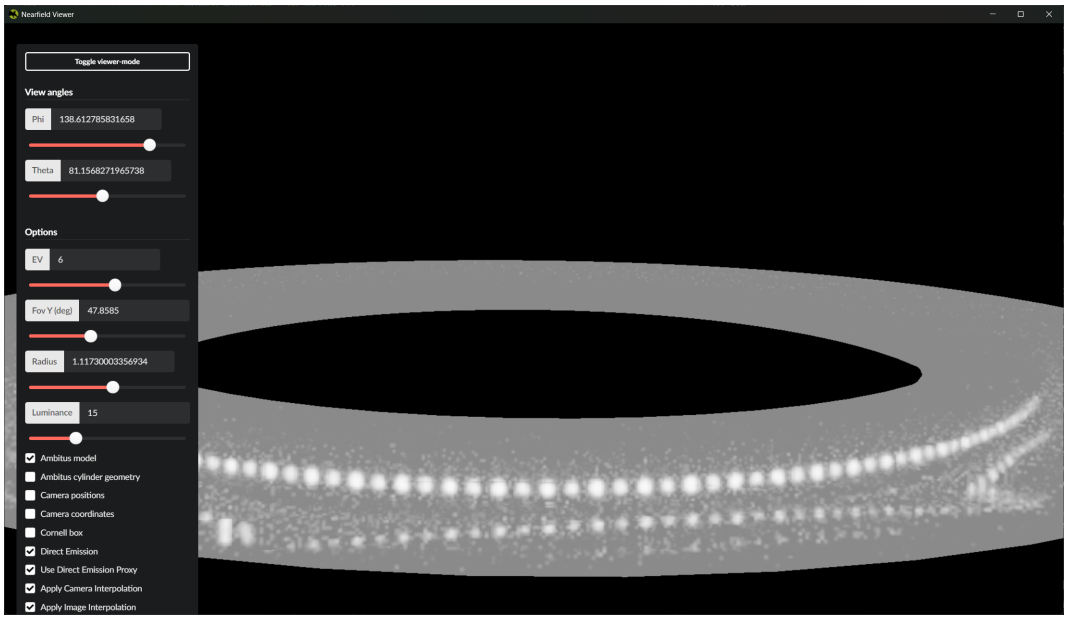


Figure 9: Ambitus ray images rendered using the geometry-aware method. The duplication artifacts are no longer visible compared to Fig. 7c.

6 Extension: Geometry-aware reprojection

Our baseline method treats every measured image as if it is located on the capture sphere (infinite-depth billboard), causing view-dependent misalignment. A simple fix is possible if the luminaire's geometry is known. We therefore added a geometry-aware variant, where each shaded pixel first intersects a proxy of the luminaire at point P. Then, to sample the ray image, we use the direction from the selected camera to the surface point P. This corrects the parallax effect greatly, as evident in Fig. 9.

7 Future Work

Although our ray image viewer renders the Ambitus scan in real time, several improvements remain open. First, the CPU preprocessing time could be shortened by parallelizing the CSR construction. Second, higher-resolution ray images are available, but they are too large for current GPU memory. A neural network trained on the ray data, similar to the approach of Zhu et al. [8], could significantly reduce the memory footprint to the point that it could reside on GPU memory [8]. This would open opportunities for GPU-accelerated light simulations of complex luminaires directly from near-field data.

8 Conclusion

We have presented the first real-time viewer that renders the luminaire itself with luminance values projected from the raw data of a near-field measurement. By losslessly compressing ray images in CSR format and re-projecting the correct ray image for each screen pixel, our viewer sustains 60 fps at Full HD, with about 8 minutes of preprocessing time. Still, limitations such as large memory requirements and interpolation artifacts require further research. Overall, the viewer shows that near-field data are viable for interactive light design work and provides a baseline for future neural or analytical methods that reduce the memory footprint and improve visual quality.

References

[1] J Quintero, Stefaan Forment, and Peter Hanselaer. In: *Proceedings of CIE Expert Symposium on Spectral and Imaging Methods for Photometry and Radiometry*. CIE CENTRAL BUREAU. 2010, pp. 9–13.

[2] G. M. Dotreppe et al. "Off-axis limiting photometric distance of Lambertians and narrow beams". In: *Lighting Research and Technology* 56 (6 Oct. 2024), pp. 550–569. ISSN: 14770938. DOI: 10.1177/14771535231208933.

[3] V. Jacobs et al. "Near-field and far-field goniophotometry of narrow-beam LED arrays". In: *Lighting Research and Technology* 47 (4 June 2015), pp. 470–482. ISSN: 14770938. DOI: 10.1177/1477153514530139.

[4] Sebastian Häring. "Erweiterung des Simulationsprozesses von Lichtfeldern für die virtuelle Leuchtenentwicklung im Automobilbau". PhD thesis. 2009. URL: https://www.db-thueringen.de/receive/dbt_mods_00014579.

[5] Henrik Wann Jensen. "Global Illumination using Photon Maps". In: *Rendering Techniques '96: Proceedings of the Eurographics Workshop in Porto, Portugal, June 17–19, 1996*. Ed. by Xavier Pueyo and Peter Schröder. Eurographics. Vienna: Springer Vienna, 1996, pp. 21–30. ISBN: 978-3-211-82883-0. URL: https://link.springer.com/book/10.1007/978-3-7091-7484-5.

[6] Niklas Smal and Maksim Aizenshtein. "Real-Time global illumination with photon mapping". In: *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress Media LLC, Feb. 2019, pp. 409–436. ISBN: 9781484244272. DOI: 10.1007/978-1-4842-4427-2_24.

[7] Edgar Velázquez-Armendáriz et al. "Complex luminaires: Illumination and appearance rendering". In: *ACM Transactions on Graphics* 34 (3 Apr. 2015). ISSN: 15577368. DOI: 10.1145/2714571.

[8] Junqiu Zhu et al. “Neural complex luminaires: Representation and rendering”. In: *ACM Transactions on Graphics* 40 (4 July 2021). ISSN: 15577368. DOI: 10.1145/3450626.3459798.

[9] Zumtobel Lighting GmbH. *Ambitus – Product Page*. 2025. URL: https://www.zumtobel.com/com-en/products/ambitus.html (visited on 08/22/2025).

[10] TechnoTeam Bildverarbeitung GmbH. *RIGO 801 Goniophotometer*. 2025. URL: https://www.technoteam.de/produkte/goniophotometer_rigo801/index_ger.html (visited on 08/22/2025).

[11] Erik Reinhard et al. “Photographic tone reproduction for digital images”. In: *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 2023, pp. 661–670. DOI: 10.1145/3596711.3596781.

[12] VRVis. *Aardvark Platform Page*. 2025. URL: https://aardvarkians.com/ (visited on 08/22/2025).

[13] TechnoTeam Bildverarbeitung GmbH. *TechnoTeam Converter product page:* 2025. URL: https://www.technoteam.de/products/goniophotometer_rigo801/rigo801_software/converter_801/index_eng.html (visited on 08/22/2025).