PowerGS: Display-Rendering Power Co-Optimization for Foveated Radiance-Field Rendering in Power-Constrained XR Systems: Supplementary Material

WEIKAI LIN, University of Rochester, USA
SUSHANT KONDGULI, Reality Labs Research, Meta, USA
CARL MARSHALL, Reality Labs Research, Meta, USA
YUHAO ZHU, University of Rochester, USA

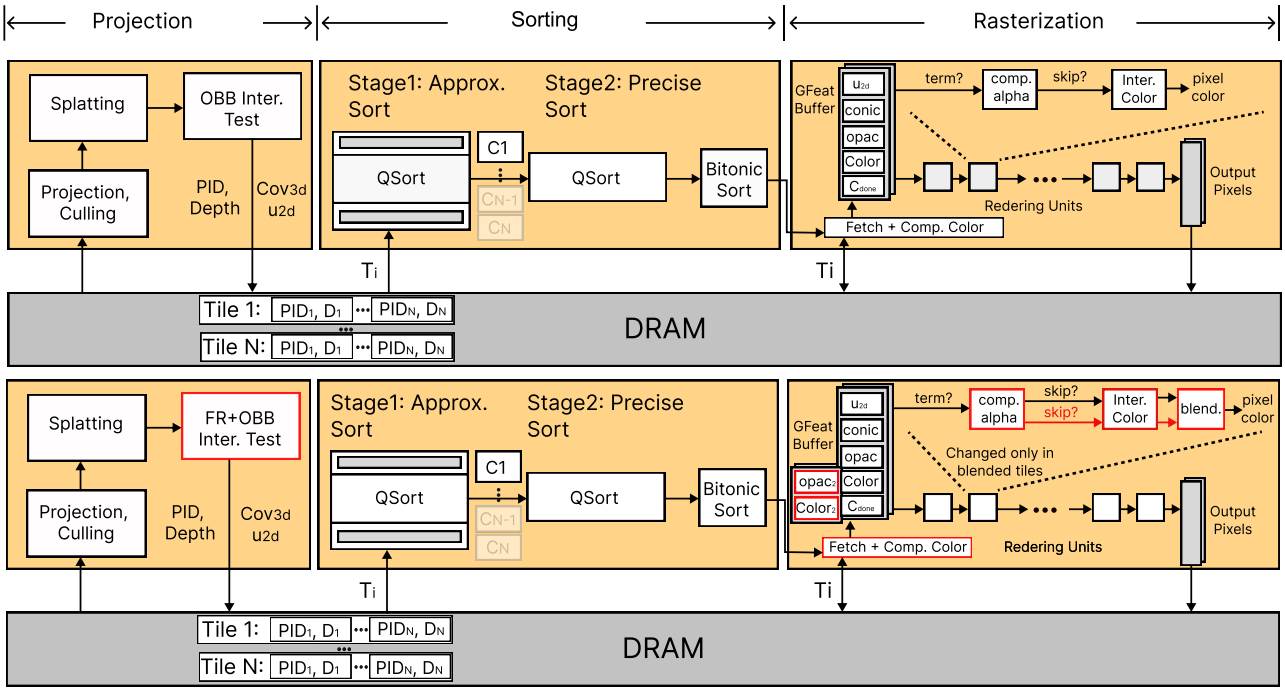

Fig. 1. Hardware models used in our rendering power estimation. (top): the original GSCore hardawre [Lee et al. 2024]; (bottom): GSCore augmented to support foveated rendering, where the augmentations are highlighted.

A Rendering Power Modeling

A.1 Unit Energy

Unit energy depends on specific hardware implementation and fabrication process. For unit FLOP energy, we use the number from a 16-bit floating-point (FP16) Multiply-Accumulate (MAC) design [Tortorella et al. 2022]. FP16 is shown to be sufficiently accurate for 3DGS [Lee et al. 2024]. SRAM unit energy (energy per Byte) is estimated by compiling a 256KB SRAM using the Arm Artisan memory compiler [Arm 2024], which matches the size of GSCore [Lee et al. 2024], a state-of-the-art 3DGS hardware accelerator. DRAM unit energy (energy per Byte) is estimated based on a 16 Gb LPDDR5x design [Ahn et al. 2024]. FLOP and SRAM energy numbers are scaled to an advanced fabrication process node (7nm) using the DeepScale tool [Sarangi and Baas 2021]. The process node is representative of that used in today's high-end integrated circuits. Overall, the energy per FLOP is 0.53 pJ, SRAM access consumes 0.24 pJ per byte, and DRAM access consumes 10.88 pJ per byte in our modeling.

A.2 Operation Count for Uniform Rendering

Estimating the operation counts required to run 3DGS $\mathcal{P}$ at pose T involves two steps: modeling a hardware architecture to support 3DGS and executing a 3DGS model at pose T to collect the related statistics: the total number of FLOPS, the total Bytes of SRAM accesses, and the total Bytes of DRAM accesses. Our hardware architecture is based on GSCore [Lee et al. 2024], as illustrated in Fig. 1 (top), and is divided into three stages: Projection, Sorting, and Rasterization.

Authors' Contact Information: Weikai Lin, University of Rochester, USA, wlin33@ur.rochester.edu; Sushant Kondguli, Reality Labs Research, Meta, USA, sushantkondguli@meta.com; Carl Marshall, Reality Labs Research, Meta, USA, csmarshall@meta.com; Yuhao Zhu, University of Rochester, USA, yzhu@rochester.edu.

In the Projection stage, three modules are considered: frustum culling, splatting, and tile-point intersection test based on the Oriented Bounding Box (OBB). FLOPs and data access patterns of first two modules are same as 3DGS official code [Kerbl et al. 2023], OBB test is based on GSCore paper. We collect input and output volumes for these modules and convert the collected data into FLOPs and memory access counts.

In the Sorting stage, points intersecting each tile are sorted from near to far. While 3DGS official code uses GPU-based RadixSort, we model a piece of dedicated sorting hardware for sorting. The sorting units sort the points hierarchically, as described in GSCore. This design includes two 7-pivot quicksort unit and a 16-channel bitonic sorter. The operation counts for this stage are obtained by collecting the points-tile intersection statistics.

In the Rasterization stage, point attributes are fetched from DRAM and stored in an on-chip SRAM feature buffer for alpha blending. Rasterization units access attributes from SRAM to compute alpha, point color, and integrate the color into pixels. The read attributes are reused across units in a 1-D systolic array manner to minimize on-chip buffer access. Finally, the computed pixel colors are written back to DRAM. Computations in this stage follow the same algorithm as 3DGS official code. By collecting input and output volumes of each module, we can infer the FLOPs and memory access counts.

The total operation counts (FLOPs and number of SRAM/DRAM accesses) for a frame is obtained by summing the results from all three stages. Multiplying the operation counts by the unit energy provides the per-frame energy.

A.3 Supporting Foveated Rendering

We also extend GSCore to support computation-sharing foveated rendering, as shown in Fig. 1 (bottom), where the extensions are highlighted. The key difference lies in the Projection stage, where we extend the OBB test with eccentricity-based foveated filtering. This module takes user's gaze as input, computes the eccentricity of points using their positions in 2D image space, and filters out points not needed at the current eccentricity level.

As with all prior foveated rendering techniques, we blend different levels at the boundaries by rendering pixel colors for boundary tiles twice (one for each level) and blending them. The blending is done at the end of the Rasterization stage. The blending introduce extra overhead. Fortunately, due to our computation-sharing strategy, around 79% computations and 66% memory access in Rasterization can be shared across levels.

B Iso-Quality Curve Reconstruction

B.1 Implementation Details

Normalization. We reconstruct the underlying display power model $D(\rho)$ and rendering power model $R(\rho)$ by fitting an inverse Michaelis–Menten kinetics from samples. We find at most 5 samples are usually sufficient.

We also find that normalizing the display/rendering power and pruning ratio of the samples to [0,1] before fitting the model parameters improves the regression stability. The normalization is done such that the maximum and minimum power/ρ value in the samples is cast to 1 and 0, respectively, and other values are scaled accordingly:

$$\tilde{x} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{1}$$

where x_{min} and x_{max} are the minimum and maximum values in the samples, x is the unnormalized value, and $\tilde{x}$ is the normalized value. The inverted normalization is applied to the model output to obtain the actual power value.

Solving for ρ . To solve for the optimal ρ, we seek the value of ρ that minimizes $G(\rho) = D(\rho) + R(\rho)$, where $G(\rho)$ represents the total power. We can prove that within the domain $[0, 1]$, the inverse Michaelis–Menten functions are convex, and (inverse) normalization does not affect their convexity (see Sec. B.2). Therefore, the sum $G(\rho)$ is also convex.

To find the minimum, we solve $\frac{dG(\rho)}{d\rho} = 0$ using `sympy` [Meurer et al. 2017] for an analytical solution. The closed-form expression for optimal pruning ratio ρ_{opt} is shown below:

$$\Delta\rho = \rho_{\max} - \rho_{\min},\ \Delta r = r_{\max} - r_{\min},\ \Delta d = d_{\max} - d_{\min}$$

$$A = \sqrt{K_r K_d V_r V_d}\,\sqrt{\Delta r\,\Delta d}\,(K_r + K_d + 1)\,\Delta\rho$$

$$B = -K_r K_d\,\Delta\rho\,\left[V_r \Delta r + V_d \Delta d\right]$$

$$C = K_r V_r \rho_{\min}\Delta r \;-\; K_d V_d \rho_{\max}\Delta d$$

$$\rho_{\text{opt}} = \frac{\pm\left(A + B + C\right)}{K_r V_r\,\Delta r \;-\; K_d V_d\,\Delta d}$$

where $\rho_{\max}$ and $\rho_{\min}$ are the maximum and minimum sampled pruning ratios. $r_{\max}, r_{\min}$ and $d_{\max}, d_{\min}$ correspond to the power bounds: r refers to rendering power, and d to display power. K_r, V_r, K_d, V_d are Michaelis–Menten parameters fitted to the rendering and display power curves, respectively. Note that although there are two possible ρ_{opt}, they are negations of each other. Therefore, at most one of them can lie within the valid range $[\rho_{\min}, \rho_{\max}]$, within which the total power function $G(\rho)$ is convex (see proof in Sec. B.2).

If both ρ_{opt} lie outside the domain $[\rho_{\min}, \rho_{\max}]$, it means that the minimum of the total power function occurs at the boundary. In this case, we evaluate the boundary values $\rho = \rho_{\min}$ and $\rho = \rho_{\max}$, and select the one with lower total power. This ensures that all evaluations remain within the supported domain, thereby performing interpolation rather than extrapolation on the models.

B.2 Convexity Proof

We now prove that our inverse Michaelis–Menten function in normalized space (i.e., $x \in [0, 1]$) is convex. We begin with the standard Michaelis–Menten function:

$$\text{MM}(x) = \frac{Vx}{K + x},$$

where $V, K > 0$ and $x \geq 0$. The rendering power function $\mathcal{R}(x)$ takes its y-inverse form, which is defined by

$$\mathcal{R}(x) = 1 - \text{MM}(x) = 1 - \frac{Vx}{K + x}.$$

To establish that $\mathcal{R}(x)$ is convex on $[0,1]$, we compute its second derivative. First, the first derivative is

$$\mathcal{R}'(x) = \frac{-VK}{(K+x)^2}.$$

Then, the second derivative becomes

$$\mathcal{R}''(x) = -VK \times \left(-2\,(K+x)^{-3}\right) = 2\,\frac{VK}{(K+x)^3}.$$

Since $V, K > 0$ and $x \geq 0$, it follows that

$$\mathcal{R}''(x) = 2 \cdot \frac{VK}{(K+x)^3} > 0 \quad \text{for all } x \geq 0.$$

A strictly positive second derivative implies that $\mathcal{R}(x)$ is strictly convex on $[0,\infty)$, so it must be convex in $[0,1]$.

The display power function $\mathcal{D}(x)$ takes the x-inverse form of the y-inverse Michaelis-Menten function above:

$$\mathcal{D}(x) \;=\; 1 - \mathrm{MM}(1-x) \;=\; 1 - \frac{V(1-x)}{K+(1-x)}.$$

To show $\mathcal{D}(x)$ is convex for x in $[0,1]$, we examine its second derivative.

$$\mathcal{D}'(x) = \frac{VK}{(K+1-x)^2},$$

$$\mathcal{D}''(x) = VK \times 2\,(K+1-x)^{-3} = \frac{2VK}{(K+1-x)^3}.$$

Since $V, K > 0$, it follows that

$$\mathcal{D}''(x) \;=\; \frac{2VK}{(K+1-x)^3} \;>\; 0 \quad \text{for all } x \in [0, K+1).$$

Since $K > 0$, $\mathcal{D}(x)$ is convex within $[0,1]$.

C Extending PowerGS to Support Foveated Rendering

Uniformly rendering high-quality content across the entire FoV is unnecessary for XR applications, as human visual acuity decreases with eccentricity. Foveated Rendering (FR) exploits this limitation to render content with varying quality depending on eccentricity [Guenter et al. 2012; Patney et al. 2016].

We assume a FR paradigm proposed in Lin et al. [2024, 2025], where, as with all prior FR work, the FoV is divided into multiple quality regions (R_1 having the highest quality), each covering a range of eccentricities with a different quality constraint, shown in Fig. 2(A). Like much of the prior FR work on radiance-field rendering [Deng et al. 2022; Franke et al. 2024], RTGS renders each quality region using a separate 3DGS model pruned from a dense model. As a result, our joint power optimization procedure can be applied to each quality region independently. This section describes the implementation details on how PowerGS is integrated into RTGS to support FR.

C.1 Basic Idea

Fig. 2 illustrates the basic idea of RTGS and how our PowerGS framework is integrated to support FR. In the basic multi-model FR paradigm [Deng et al. 2022; Franke et al. 2024], the models used in different regions are trained/pruned independently from the same dense model, so the total rendering power is the sum of that of each level of model. Instead, RTGS allow models of different regions to share parameters and, thus, computation, so that the total rendering power is lower than that of the sum. This is shown in Fig. 2(B), where colored ellipses denote those in a higher-quality model R_n that are absent in a lower-quality R_{n+1} model.

Fig. 2(C) shows how PowerGS is used to obtain models of different regions. We start with a dense model and prune it to obtain the model for region 1 (R_1), which is then used as the starting point for pruning to obtain the model for region 2 (R_2). This process is repeated for subsequent regions. This way, points used by a lower quality model form a strict subset of those used by a higher quality model. Each pruning step applies the PowerGS method described in Section 4 of the main text, i.e., obtaining the iso-quality curve through sample-and-reconstruction and solving the power optimization problem.

When pruning the model of each quality region, RTGS uses a quality measure inspired by the ventral metamerism in the visual cortex [Freeman and Simoncelli 2011], which computes the overall statistics (e.g., mean and standard deviation) within retinal spatial pooling areas to assess the similarities between two visual stimuli (e.g., a foveated image and a reference image). The pooling area grows with eccentricity, accounting for the eccentricity-dependent acuity falloff. We call it the Human Vision System Quality (HVSQ) metric. The HVSQ metric is used as the quality constraint Q_{min} when solving the PowerGS power optimization problem in Eqn. 1 in the main text.

To retain flexibility for training lower-quality models, 4 out of the 59 point attributes (opacity and the DC component of spherical harmonic coefficients) are tunable when fine-tuning a lower-quality model; the remaining parameters are fixed once R_1 is trained and are shared across all quality levels.

An advantage of this partial parameter sharing is to allow different models to share all the computations up until the color integration step in rasterization, reducing rendering power. Fig. 2(D) shows the three main stages in 3DGS rendering, including projection, sorting, and rasterization, and the rasterization stage is subdivided into the pre-integration stage and the final color integration stage[1]. Up to 87.7% of the rendering power consumption (the first three stages) of a R_1 model is unrelated to opacity and SHs DC and, thus, can be shared across all models. This computation sharing is faithfully modeled in hardware and the rendering power modeling, as discussed in Sec. A.3.

C.2 Level 1 (Highest-Quality Region) Training

During the training of the fovea model (first level), we use both PSNR and SSIM as the quality metrics. PSNR ensures pixel-wise fidelity, while SSIM preserves structural details.

Specifically, we set a quality requirement (Q_{min}) for each metric, such as 99% of that of the dense model. When dynamically adjusting λ, we evaluate and monitor all metrics individually. Only when all metrics meet their respective quality requirements do we consider the quality satisfied and increase λ to trade surplus quality for lower power. Conversely, if any metric fails to meet its requirement, the quality is considered unsatisfactory, and λ is decreased to restore the quality.

[1] Projection splats 3D Gaussian points onto the image plane. Sorting arranges the points by depth. Rasterization integrates point colors into pixels. The actual color integration depends on opacity and SH coefficients that can differ between models, but all the steps before integration (e.g., Gaussian evaluation) can be shared across models.

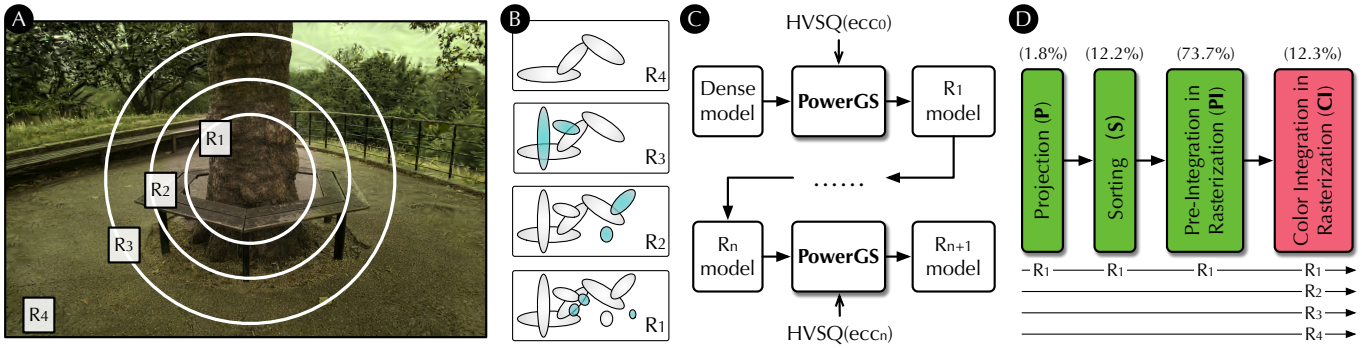

Fig. 2. Overview of how PowerGS builds on top of RTGS [Lin et al. 2024, 2025] to support FR. (**A**): as with all prior FR work, we divide the Field-of-View (FoV) into different quality regions. (**B**): each quality region is rendered by a separate model; points used by a lower quality model are strictly a subset of that of the higher quality model; color ellipses denote ellipses of a higher quality model R_n that are absent in a lower quality model R_{n+1}. (**C**): PowerGS can be easily extended to support FR: model R_{n+1} is derived from model R_n using the power optimization described in the main paper (dense model being R_0) with a ventral-metamer–inspired, eccentricity-dependent quality metric HVSQ [Freeman and Simoncelli 2011; Walton et al. 2021]. (**D**): point sharing enables computation sharing, further reducing rendering power: all the computations in the 3DGS rendering pipeline except the final color integration step can be shared across models (blending across quality regions is omitted in the figure but is implemented), and the shared portion is about 87.7% of the R_1's rendering power (numbers above each stage show the rendering power breakdown).

C.3 Lower-Quality Regions Training Details

The HVS Quality (HVSQ) loss [Walton et al. 2021, 2022] is inspired by classic neuroscience studies on the ventral pathway of the human visual system [Freeman and Simoncelli 2011]. The loss evaluates the similarity between a reference image and an altered image, incorporating the eccentricity of each pixel, which depends on the display resolution and the eye-display distance. A lower loss indicates higher similarity between the images as perceived by humans. We use the implementation of the HVS Loss in Akşit et al. [2024], which we refer interested readers to for details.

The HVSQ loss is based on the principle that the retina aggregates photoreceptor outputs into spatial regions known as spatial poolings. In image space, these spatial poolings correspond to clusters of adjacent pixels, with pooling size increasing quadratically with eccentricity. Computational models of the human visual system [Freeman and Simoncelli 2011] suggest that as long as the statistical properties (mean and standard deviation) of the content within a spatial pooling are similar between two images, humans cannot distinguish between them.

After obtaining the foveal model that minimizes power while satisfying the quality requirements, we measure its quality using HVSQ with an eccentricity of 0°. We then use the HVSQ of the foveal region as the quality requirement for all higher-level regions during pruning and fine-tuning. This ensures that the HVSQ of higher levels is not worse than that of the fovea, representing a quality alignment between different regions in PowerGS.

In PowerGS, we set the starting eccentricity of each region to be 0°, 18°, 27°, and 33°. The HVSQ for all regions is computed based on their starting eccentricity. Even though the same numerical HVSQ constraint is applied to all levels, the quality constraints for higher levels are effectively relaxed, because the HVS loss uses a larger pool area to calculate statistics in higher eccentricities.

During subjective evaluation, users are instructed to fixate at the center of the visual field. This is the same as the fixed-foveated rendering regime used in XR devices such as Meta Quest [Meta 2024]. This mitigates flickering caused by eye movements—a known challenge in FR that is beyond the scope of this paper [Arabadzhiyska et al. 2017; Patney et al. 2016].

D Additional Results

We provide experiments on two additional real-world datasets in addition to Mip-NeRF360 [Barron et al. 2022] (9 scenes) and Synthetic NeRF [Mildenhall et al. 2020] (8 scenes), including Tanks&Temples [Knapitsch et al. 2017] (2 scenes) and Deep Blending [Hedman et al. 2018] (2 scenes). We provide detailed scene-wise breakdown as described below.

Detailed Scene-wise Metrics. In Tbl. 1 and Tbl. 2, we report the per-scene metrics for seven methods: 3DGS, MiniSplatting-D, MiniSplatting, LightGS-H, FR-Display-H, FR-Render-H, and our PowerGS-H. The first two are dense 3DGS variants that offer the highest quality but incur high power consumption. The next two are pruned, efficient versions of 3DGS, but apply uniform rendering. The final three methods implement foveated rendering in different ways: FR-Display-H optimizes only for display power by color adjustment, FR-Render-H optimizes only for rendering power via pruning, and PowerGS-H co-optimizes both to minimize the total power.

Specifically, Tbl. 1 presents results on the 13 real-world scenes (9 from Mip-NeRF360, 2 from Tanks&Temples, and 2 from Deep Blending), while Tbl. 2 includes results from the 8 Synthetic NeRF scenes. For each method, we report quality metrics: PSNR, SSIM, and LPIPS. For foveated methods, these metrics are computed in the fovea region, as we align HVSQ across eccentricity. We also report the Rendering Power, Display Power, and Total Power to highlight the power savings achieved by PowerGS-H. Additionally, we report Giga Floating Point Operations (GFLOPs) per frame, which

is commonly used as a proxy for the rendering latency, which is important to real-time rendering.

The results demonstrate that PowerGS-H achieves comparable or superior visual quality to all other methods, while consistently achieving the *lowest* total power consumption across *all* scenes. At the same time, it is the second fastest among all methods (only slightly slower than FR-Render-H), achieving approximately 5× fewer GFLOPs on average compared to its dense counterpart, thanks to FR.

Scene-wise Visualizations. Fig. 3 to 6 show qualitative results for Mip-NeRF360 scenes. Fig. 7 and 8 show visualizations for Tanks&Temples and Deep Blending, respectively. Fig. 9 to Fig. 12 display qualitative comparisons for Synthetic NeRF scenes.

Each figure contains comparisons of the 7 methods, with annotations for Rendering Power and Display Power. For foveated versions, the gaze is fixed at the center of the image. We provide zoom-in views of both the foveal and peripheral regions to allow better visual comparison.

The visualization results show that all methods render high-quality outputs at the fovea. Uniform rendering consumes more power than foveated approaches, primarily due to unnecessarily high quality in the peripheral regions. Among foveated methods, optimizing for display power (FR-Display-H) introduces a greenish/yellowish tint in the periphery, while optimizing for rendering power (FR-Render-H) leads to detail loss in the periphery. PowerGS-H strikes a balance between these two extremes, achieving the lowest total power consumption while preserving perceptual quality.

References

Hyun-A Ahn, Yoo-Chang Sung, Yong-Hun Kim, Janghoo Kim, Kihan Kim, Dong-Hun Lee, Young-Gil Go, Jae-Woo Lee, Jae-Woo Jung, Yong-Hyun Kim, et al. 2024. A 1.01-V 8.5-Gb/s/pin 16-Gb LPDDR5x SDRAM With Advanced I/O Circuitry for High-Speed and Low-Power Applications. *IEEE Journal of Solid-State Circuits* (2024).

Kaan Akşit, Jeanne Beyazian, Praneeth Chakravarthula, Ziyang Chen, Mustafa Doğa Doğan, Ahmet Hamdi Güzel, Yuta Itoh, Henry Kam, Ahmet Serdar Karadeniz, Koray Kavaklı, Liang Shi, Josef Spjut, David Robert Walton, Jialun Wu, Doğa Yılmaz, Wang Yujie, Runze Zhu, Weijie Xie, and Yicheng Zhan. 2024. *Odak*. https://doi.org/10.5281/zenodo.13995841

Elena Arabadzhiyska, Okan Tarhan Tursun, Karol Myszkowski, Hans-Peter Seidel, and Piotr Didyk. 2017. Saccade landing position prediction for gaze-contingent rendering. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–12.

Arm. 2024. Arm Artisan IPs. https://www.arm.com/products/silicon-ip-physical/artisan-ip.

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*. 5470–5479.

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* 28, 11 (2022), 3854–3864.

Linus Franke, Laura Fink, and Marc Stamminger. 2024. VR-Splatting: Foveated Radiance Field Rendering via 3D Gaussian Splatting and Neural Points. *arXiv preprint arXiv:2410.17932* (2024).

Jeremy Freeman and Eero P Simoncelli. 2011. Metamers of the ventral stream. *Nature neuroscience* 14, 9 (2011), 1195–1201.

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D graphics. *ACM transactions on Graphics (tOG)* 31, 6 (2012), 1–10.

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.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.* 42, 4 (2023), 139–1.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)* 36, 4 (2017), 1–13.

Junseo Lee, Seokwon Lee, Jungi Lee, Junyong Park, and Jaewoong Sim. 2024. GSCore: Efficient Radiance Field Rendering via Architectural Support for 3D Gaussian Splatting. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 497–511.

Weikai Lin, Yu Feng, and Yuhao Zhu. 2024. RTGS: Enabling Real-Time Gaussian Splatting on Mobile Devices Using Efficiency-Guided Pruning and Foveated Rendering. *arXiv preprint arXiv:2407.00435* (2024).

Weikai Lin, Yu Feng, and Yuhao Zhu. 2025. MetaSapiens: Real-Time Neural Rendering with Efficiency-Aware Pruning and Accelerated Foveated Rendering. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 669–682.

Meta. 2024. Fixed foveated rendering (FFR). https://developers.meta.com/horizon/documentation/unity/os-fixed-foveated-rendering/.

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (2017), e103.

B Mildenhall, PP Srinivasan, M Tancik, JT Barron, R Ramamoorthi, and R Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*.

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), 1–12.

Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A tool for the accurate estimation of technology scaling in the deep-submicron era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

Yvan Tortorella, Luca Bertaccini, Davide Rossi, Luca Benini, and Francesco Conti. 2022. RedMulE: A compact FP16 matrix-multiplication accelerator for adaptive deep learning on RISC-V-based ultra-low-power SoCs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1099–1102.

David R. Walton, Rafael Kuffner Dos Anjos, Sebastian Friston, David Swapp, Kaan Akşit, Anthony Steed, and Tobias Ritschel. 2021. Beyond Blur: Ventral Metamers for Foveated Rendering. *ACM Trans. Graph. (Proc. SIGGRAPH 2021)* 40, 4 (2021).

David R Walton, Koray Kavaklı, Rafael Kuffner Dos Anjos, David Swapp, Tim Weyrich, Hakan Urey, Anthony Steed, Tobias Ritschel, and Kaan Akşit. 2022. Metameric varifocal holograms. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 746–755.

Table 1. **Quantitative Evaluation Across Three Real-World Datasets.** We evaluate seven methods on three real-world datasets using seven metrics. PowerGS-H (green) consistently achieves the lowest total power consumption across all scenes while maintaining comparable or better quality than other methods. We use medals 🥇, 🥈, and 🥉 to highlight the **1st**, **2nd**, and **3rd** places in each metric, respectively.

Metrics	Method	Mip-NeRF360									T&T		Deep Blending		Avg(13 Scenes)
		Bicycle	Bonsai	Counter	Flowers	Garden	Kitchen	Room	Stump	Treehill	Train	Truck	Playroom	Drjohnson	
PSNR↑	3DGS	25.2057	32.2180	29.0080	21.5542	27.3732	31.5089	31.5607	26.6212	22.4450	22.0455	25.3959	29.9417	29.1603	27.2337
	MiniSplatting-D	25.5416	32.0301	28.2741	21.5112	27.7232	31.4544	31.5910	27.1492	22.2636	21.1891	25.3691	30.5035	29.3389	27.2261
	MiniSplatting	25.2008	31.2495	28.4618	21.5723	26.8820	31.2216	31.2232	27.2471	22.6310	21.4615	25.2486	30.5157	29.3900	27.1004
	LightGS-H	25.1745	32.0768	28.9753	21.5501	27.3057	31.3948	31.8148	26.6022	22.4846	22.1594	25.4984	29.9108	29.2298	27.2444
	FR-Display-H	25.2142	31.8922	28.0383	21.2706	27.3849	31.1539	31.2274	26.8199	22.1883	22.0191	25.1287	29.5776	28.8727	26.9837
	FR-Render-H	25.4634	31.9826	28.2242	21.4697	27.5831	31.2855	31.3827	27.1333	22.4053	21.9186	25.3574	29.6523	28.9516	27.1392
	PowerGS-H	25.3044	32.0587	28.2237	21.2406	27.6358	31.0259	31.2981	26.9086	22.4189	20.9853	25.1358	29.9251	28.9341	27.0073
SSIM↑	3DGS	0.7651	0.9421	0.9086	0.6056	0.8661	0.9277	0.9196	0.7726	0.6325	0.8136	0.8817	0.9063	0.9003	0.8340
	MiniSplatting-D	0.7981	0.9480	0.9085	0.6425	0.8783	0.9312	0.9286	0.8053	0.6419	0.8192	0.8898	0.9079	0.9056	0.8465
	MiniSplatting	0.7725	0.9390	0.9043	0.6245	0.8477	0.9261	0.9211	0.8056	0.6525	0.8098	0.8823	0.9137	0.9089	0.8391
	LightGS-H	0.7620	0.9386	0.9020	0.6044	0.8613	0.9229	0.9187	0.7737	0.6296	0.8058	0.8824	0.9053	0.8986	0.8312
	FR-Display-H	0.7921	0.9496	0.9082	0.6378	0.8753	0.9302	0.9295	0.7961	0.6315	0.8177	0.8816	0.9034	0.8982	0.8424
	FR-Render-H	0.7917	0.9430	0.9036	0.6369	0.8705	0.9246	0.9229	0.8015	0.6379	0.8079	0.8755	0.9044	0.8997	0.8400
	PowerGS-H	0.7911	0.9439	0.9014	0.6373	0.8716	0.9209	0.9214	0.8012	0.6392	0.8211	0.8861	0.9059	0.9013	0.8417
LPIPS↓	3DGS	0.2101	0.2030	0.1995	0.3359	0.1074	0.1261	0.2182	0.2156	0.3258	0.2073	0.1471	0.2444	0.2442	0.2142
	MiniSplatting-D	0.1576	0.1736	0.1753	0.2545	0.0901	0.1159	0.1878	0.1681	0.2606	0.1810	0.0999	0.2026	0.2181	0.1758
	MiniSplatting	0.2242	0.1996	0.1986	0.3274	0.1498	0.1292	0.2120	0.1984	0.3141	0.2225	0.1385	0.2365	0.2406	0.2147
	LightGS-H	0.2153	0.2107	0.2130	0.3431	0.1153	0.1375	0.2235	0.2179	0.3359	0.2285	0.1470	0.2502	0.2496	0.2221
	FR-Display-H	0.1585	0.1689	0.1724	0.2537	0.0907	0.1159	0.1818	0.1734	0.2603	0.2043	0.1455	0.2460	0.2452	0.1859
	FR-Render-H	0.1714	0.1946	0.1921	0.2775	0.1029	0.1318	0.2092	0.1943	0.3130	0.2317	0.1642	0.2596	0.2542	0.2074
	PowerGS-H	0.1717	0.1929	0.1988	0.2775	0.1016	0.1386	0.2124	0.1944	0.3177	0.2086	0.1214	0.2423	0.2453	0.2018
Rendering Power (W)↓	3DGS	0.6357	0.2603	0.3301	0.3289	0.6922	0.5195	0.3141	0.3959	0.3497	0.1835	0.2906	0.2515	0.3276	0.3754
	MiniSplatting-D	0.7099	0.6713	0.8975	0.5715	0.8469	0.9959	0.6574	0.5937	0.6110	0.4755	0.6008	0.3676	0.4182	0.6475
	MiniSplatting	0.1254	0.1665	0.2175	0.1285	0.1437	0.2238	0.1591	0.1439	0.1328	0.0674	0.0813	0.0846	0.0856	0.1354
	LightGS-H	0.2757	0.1634	0.2091	0.1826	0.3407	0.2762	0.1889	0.2333	0.2059	0.0871	0.1497	0.1296	0.1548	0.1998
	FR-Display-H	0.7152	0.6602	0.8555	0.6122	0.8735	0.9759	0.6313	0.6115	0.6314	0.2018	0.3026	0.2381	0.3089	0.5860
	FR-Render-H	0.1446	0.1094	0.1452	0.1473	0.1695	0.1760	0.0971	0.1062	0.0857	0.0510	0.0608	0.0453	0.0568	0.1073
	PowerGS-H	0.1547	0.1222	0.1414	0.1743	0.1821	0.1512	0.1000	0.1137	0.1044	0.0776	0.1127	0.0643	0.0674	0.1205
Display Power (W)↓	3DGS	0.2688	0.3586	0.3301	0.4255	0.4799	0.6624	0.3184	0.1531	0.3922	0.8296	1.0368	0.7138	0.4176	0.4913
	MiniSplatting-D	0.2666	0.3572	0.3307	0.4355	0.4780	0.6629	0.3187	0.1520	0.4012	0.8287	1.0371	0.7121	0.4147	0.4920
	MiniSplatting	0.2660	0.3562	0.3296	0.4278	0.4779	0.6641	0.3189	0.1517	0.3924	0.8364	1.0352	0.7123	0.4149	0.4910
	LightGS-H	0.2686	0.3572	0.3310	0.4228	0.4789	0.6613	0.3197	0.1536	0.3880	0.8278	1.0361	0.7162	0.4168	0.4906
	FR-Display-H	0.1951	0.3074	0.2729	0.2876	0.3905	0.5985	0.2801	0.1094	0.2348	0.6434	0.8467	0.6441	0.3604	0.3978
	FR-Render-H	0.2646	0.3586	0.3323	0.4161	0.4794	0.6624	0.3172	0.1519	0.3789	0.8330	1.0285	0.7130	0.4165	0.4886
	PowerGS-H	0.2099	0.3237	0.2750	0.3020	0.4163	0.6000	0.2975	0.1164	0.2428	0.6421	0.8484	0.6530	0.3718	0.4076
GFLOPs (per frame)↓	3DGS	12.3251	5.7453	7.2813	6.2892	13.5036	11.1315	7.0784	7.7448	6.8303	3.6167	5.4464	5.5352	7.0858	7.6626
	MiniSplatting-D	13.2884	13.8290	18.1651	10.7232	16.0238	20.6333	13.8494	11.3496	11.6279	8.3948	10.5551	7.7574	8.7191	12.6859
	MiniSplatting	2.6911	3.8990	4.9697	2.6465	3.0791	5.1270	3.8121	3.0199	2.8073	1.3739	1.6785	2.0092	1.9890	3.0079
	LightGS-H	5.6120	3.7998	4.8016	3.6985	6.9012	6.1402	4.4345	4.8211	4.2665	1.8002	2.9560	3.0141	3.5305	4.2905
	FR-Display-H	13.1018	13.3774	16.8779	10.8867	15.8291	19.4941	13.1827	11.2809	11.5893	3.6941	5.5488	5.3727	6.7571	11.3071
	FR-Render-H	2.3810	2.4681	3.0816	2.3071	2.7730	3.6614	2.2709	1.8910	1.5024	1.0360	1.1914	1.1283	1.3070	2.0769
	PowerGS-H	2.6545	2.8115	3.2594	2.8318	3.1051	3.3934	2.4244	2.1055	2.0115	1.4329	2.0318	1.5710	1.5936	2.4020
Total Power (W)↓	3DGS	0.9045	0.6189	0.6602	0.7544	1.1721	1.1819	0.6325	0.5490	0.7419	1.0131	1.3275	0.9652	0.7452	0.8667
	MiniSplatting-D	0.9765	1.0285	1.2281	1.0071	1.3250	1.6588	0.9761	0.7458	1.0122	1.3043	1.6379	1.0798	0.8328	1.1394
	MiniSplatting	0.3914	0.5227	0.5471	0.5562	0.6216	0.8878	0.4780	0.2957	0.5252	0.9038	1.1165	0.7969	0.5005	0.6264
	LightGS-H	0.5443	0.5206	0.5401	0.6053	0.8196	0.9375	0.5086	0.3869	0.5940	0.9149	1.1857	0.8457	0.5716	0.6904
	FR-Display-H	0.9103	0.9676	1.1284	0.8998	1.2640	1.5744	0.9114	0.7209	0.8662	0.8452	1.1493	0.8822	0.6693	0.9838
	FR-Render-H	0.4092	0.4680	0.4775	0.5634	0.6489	0.8385	0.4144	0.2582	0.4646	0.8840	1.0893	0.7583	0.4733	0.5960
	PowerGS-H	0.3645	0.4459	0.4165	0.4763	0.5984	0.7512	0.3974	0.2301	0.3472	0.7197	0.9611	0.7173	0.4392	0.5281

Table 2. **Quantitative Evaluation Across Synthetic NeRF Dataset.** We evaluate seven methods on Synthetic NeRF Dataset using seven metrics. PowerGS-H (green) consistently achieves the lowest total power consumption across all scenes while maintaining comparable or better quality than other methods. We use medals 🥇, 🥈, and 🥉 to highlight the **1st**, **2nd**, and **3rd** places in each metric, respectively.

Metrics	Method	Synthetic NeRF								Average
		Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	(8 scenes)
PSNR↑	3DGS	🥇 35.5277	🥇 26.2770	🥇 35.4939	🥇 38.0307	🥉 36.0776	🥇 30.4911	🥇 36.6839	🥇 31.6882	🥇 33.7837
	MiniSplatting-D	34.3225	25.6544	27.4828	🥉 37.9712	🥈 36.1678	29.1815	32.9624	30.1773	31.7400
	MiniSplatting	34.1229	25.7537	26.4539	🥈 37.9958	🥇 36.4631	29.1610	32.8124	30.6274	31.6738
	LightGS-H	34.8312	🥈 26.1480	🥈 35.4556	36.5692	35.0414	29.5123	34.8355	29.6761	32.7587
	FR-Display-H	35.1196	26.0002	🥉 35.1695	37.6515	35.6891	🥉 30.1998	36.3621	31.3548	33.4433
	FR-Render-H	🥈 35.5149	🥉 26.0636	35.1627	37.7877	35.8032	🥈 30.2505	🥈 36.5323	🥉 31.4113	🥈 33.5658
	PowerGS-H	🥉 35.3227	26.0174	35.1034	37.6577	35.6006	30.1214	🥉 36.3719	🥈 31.4142	🥉 33.4512
SSIM↑	3DGS	🥇 0.9877	🥇 0.9548	🥇 0.9870	🥉 0.9853	🥉 0.9827	🥇 0.9604	🥇 0.9925	🥇 0.9063	🥇 0.9696
	MiniSplatting-D	0.9866	0.9479	0.9606	🥈 0.9860	🥈 0.9832	0.9516	0.9826	0.8878	0.9608
	MiniSplatting	0.9866	0.9503	0.9597	🥇 0.9864	🥇 0.9853	0.9533	0.9842	0.8979	0.9630
	LightGS-H	0.9851	🥉 0.9530	🥈 0.9867	0.9784	0.9795	0.9559	0.9894	0.8886	0.9646
	FR-Display-H	🥉 0.9871	🥈 0.9539	🥉 0.9865	0.9850	0.9823	🥈 0.9601	🥈 0.9925	🥈 0.9054	🥈 0.9691
	FR-Render-H	🥈 0.9873	0.9512	0.9853	0.9841	0.9816	🥉 0.9577	0.9919	🥉 0.9032	🥉 0.9678
	PowerGS-H	0.9868	0.9521	0.9850	0.9835	0.9809	0.9567	🥉 0.9921	0.9030	0.9675
LPIPS↓	3DGS	🥇 0.0105	🥇 0.0367	🥇 0.0118	🥉 0.0199	🥉 0.0160	🥇 0.0369	🥈 0.0064	🥉 0.1059	🥇 0.0305
	MiniSplatting-D	0.0117	0.0433	0.0409	🥇 0.0166	🥈 0.0127	0.0411	0.0186	🥇 0.0971	0.0353
	MiniSplatting	0.0122	0.0416	0.0364	🥈 0.0175	🥇 0.0126	0.0432	0.0148	0.1066	0.0356
	LightGS-H	0.0128	🥉 0.0388	🥈 0.0120	0.0308	0.0209	0.0462	0.0089	0.1276	0.0373
	FR-Display-H	🥈 0.0109	🥈 0.0378	🥉 0.0121	0.0201	0.0160	🥈 0.0374	🥇 0.0064	🥈 0.1051	🥈 0.0307
	FR-Render-H	🥉 0.0114	0.0418	0.0135	0.0233	0.0176	🥉 0.0402	0.0071	0.1136	🥉 0.0336
	PowerGS-H	0.0123	0.0402	0.0138	0.0247	0.0187	0.0413	🥉 0.0069	0.1136	0.0340
Rendering Power (W)↓	3DGS	0.1503	0.1359	0.1142	0.0828	0.1245	0.0689	0.0796	0.1364	0.1116
	MiniSplatting-D	0.6358	0.6689	0.6163	0.6846	0.6769	0.6555	0.6132	0.8349	0.6733
	MiniSplatting	🥉 0.0430	0.0614	0.0647	0.0561	0.0733	0.0639	0.0747	0.1045	0.0677
	LightGS-H	0.0590	🥉 0.0536	🥉 0.0416	🥉 0.0422	🥉 0.0495	🥉 0.0323	🥉 0.0336	🥉 0.0593	🥉 0.0464
	FR-Display-H	0.1502	0.1347	0.1132	0.0802	0.1208	0.0677	0.0767	0.1349	0.1098
	FR-Render-H	🥈 0.0237	🥇 0.0146	🥇 0.0111	🥈 0.0213	🥈 0.0312	🥇 0.0197	🥇 0.0111	🥇 0.0245	🥇 0.0196
	PowerGS-H	🥇 0.0228	🥈 0.0198	🥈 0.0111	🥇 0.0204	🥇 0.0303	🥈 0.0225	🥈 0.0127	🥈 0.0274	🥈 0.0209
Display Power (W)↓	3DGS	0.2085	0.1391	0.0448	0.1623	0.1878	0.0914	0.0395	0.0708	0.1180
	MiniSplatting-D	0.2077	🥉 0.1362	🥈 0.0358	🥉 0.1619	🥉 0.1866	0.0891	0.0367	0.0688	0.1153
	MiniSplatting	0.2075	0.1362	🥇 0.0350	0.1619	0.1876	0.0893	🥉 0.0364	🥉 0.0685	🥉 0.1153
	LightGS-H	0.2085	0.1391	0.0448	0.1625	0.1875	0.0906	0.0394	0.0695	0.1177
	FR-Display-H	🥇 0.1805	🥇 0.0945	🥉 0.0358	🥇 0.1464	🥇 0.1665	🥇 0.0636	🥇 0.0300	🥇 0.0518	🥇 0.0962
	FR-Render-H	🥉 0.2069	0.1395	0.0452	0.1630	0.1873	🥉 0.0886	0.0396	0.0709	0.1176
	PowerGS-H	🥈 0.1837	🥈 0.0975	0.0377	🥈 0.1495	🥈 0.1688	🥈 0.0666	🥈 0.0316	🥈 0.0560	🥈 0.0989
GFLOPs (per frame)↓	3DGS	2.8091	2.6453	2.4253	1.6929	2.4064	1.4182	1.6040	2.9816	2.2479
	MiniSplatting-D	10.2933	10.9503	11.7166	11.4359	10.8114	11.3392	10.0073	14.9711	11.4406
	MiniSplatting	🥉 0.8460	1.3287	1.5299	1.2064	1.3850	1.3758	1.7845	2.2697	1.4657
	LightGS-H	1.1143	🥉 1.0602	🥉 0.8768	🥉 0.9123	🥉 0.9791	🥉 0.6885	🥉 0.6940	🥉 1.3408	🥉 0.9583
	FR-Display-H	2.8293	2.6946	2.4641	1.7321	2.4192	1.4626	1.6132	3.0483	2.2829
	FR-Render-H	🥇 0.4533	🥇 0.2956	🥇 0.2177	🥈 0.4785	🥇 0.6295	🥇 0.4304	🥇 0.2139	🥇 0.5399	🥇 0.4073
	PowerGS-H	🥈 0.4705	🥈 0.4183	🥈 0.2307	🥇 0.4748	🥈 0.6434	🥈 0.5229	🥈 0.2538	🥈 0.6181	🥈 0.4541
Total Power (W)↓	3DGS	0.3587	0.2750	0.1589	0.2451	0.3123	0.1604	0.1191	0.2073	0.2296
	MiniSplatting-D	0.8436	0.8051	0.6521	0.8464	0.8635	0.7446	0.6499	0.9038	0.7886
	MiniSplatting	🥉 0.2506	0.1976	0.0997	0.2180	0.2609	0.1533	0.1111	0.1730	0.1830
	LightGS-H	0.2675	🥉 0.1927	🥉 0.0864	🥉 0.2048	🥉 0.2369	🥉 0.1228	🥉 0.0730	🥉 0.1289	🥉 0.1641
	FR-Display-H	0.3307	0.2292	0.1490	0.2267	0.2873	0.1313	0.1068	0.1867	0.2060
	FR-Render-H	🥈 0.2306	🥈 0.1541	🥈 0.0563	🥈 0.1844	🥈 0.2185	🥈 0.1082	🥈 0.0507	🥈 0.0955	🥈 0.1373
	PowerGS-H	🥇 0.2065	🥇 0.1174	🥇 0.0488	🥇 0.1699	🥇 0.1991	🥇 0.0891	🥇 0.0443	🥇 0.0833	🥇 0.1198

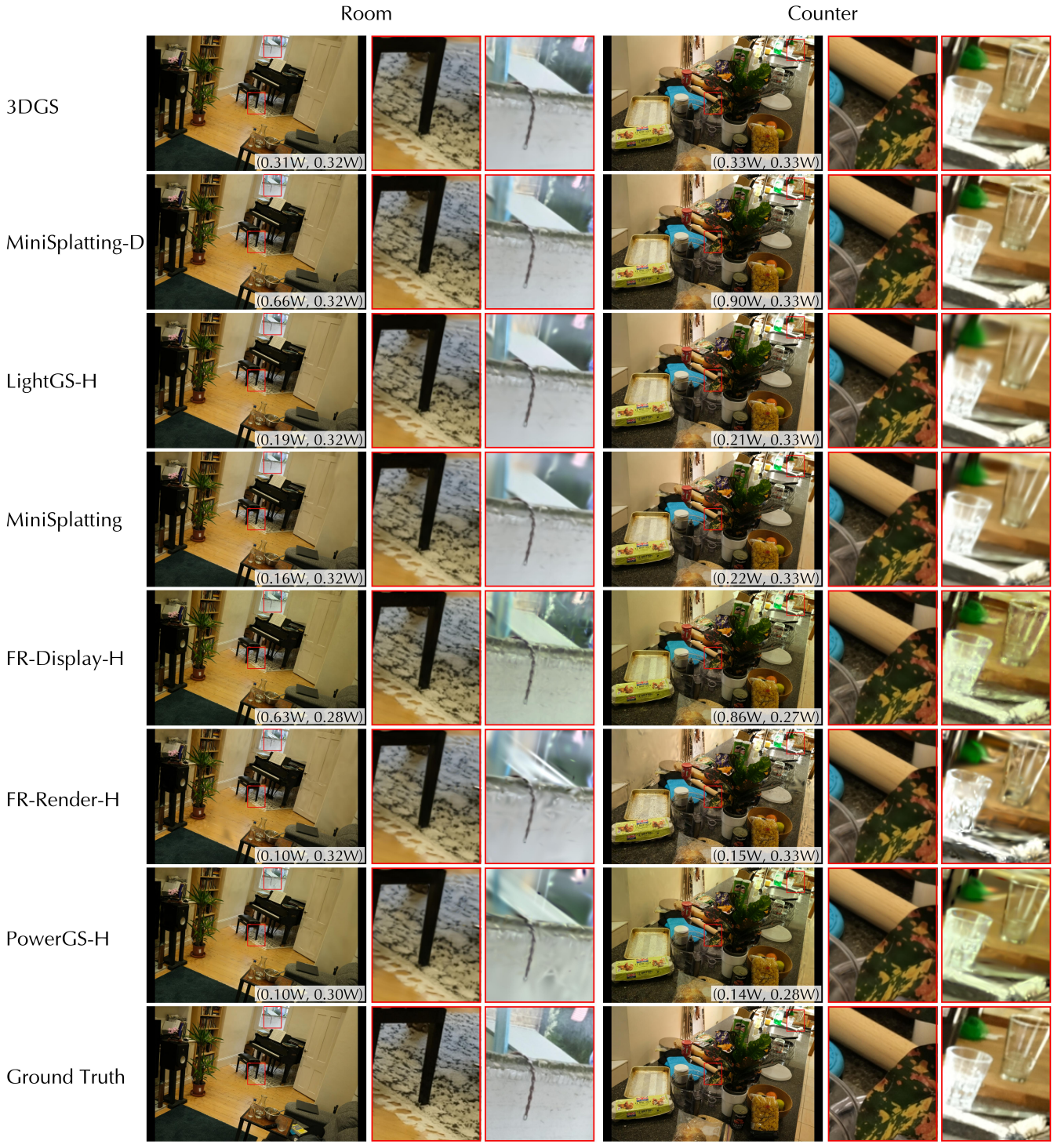

Fig. 3. **Qualitative Evaluation.** Rendered images of two scenes in Mip-NeRF360 are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

Fig. 4. **Qualitative Evaluation.** Rendered images of two scenes in Mip-NeRF360 are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

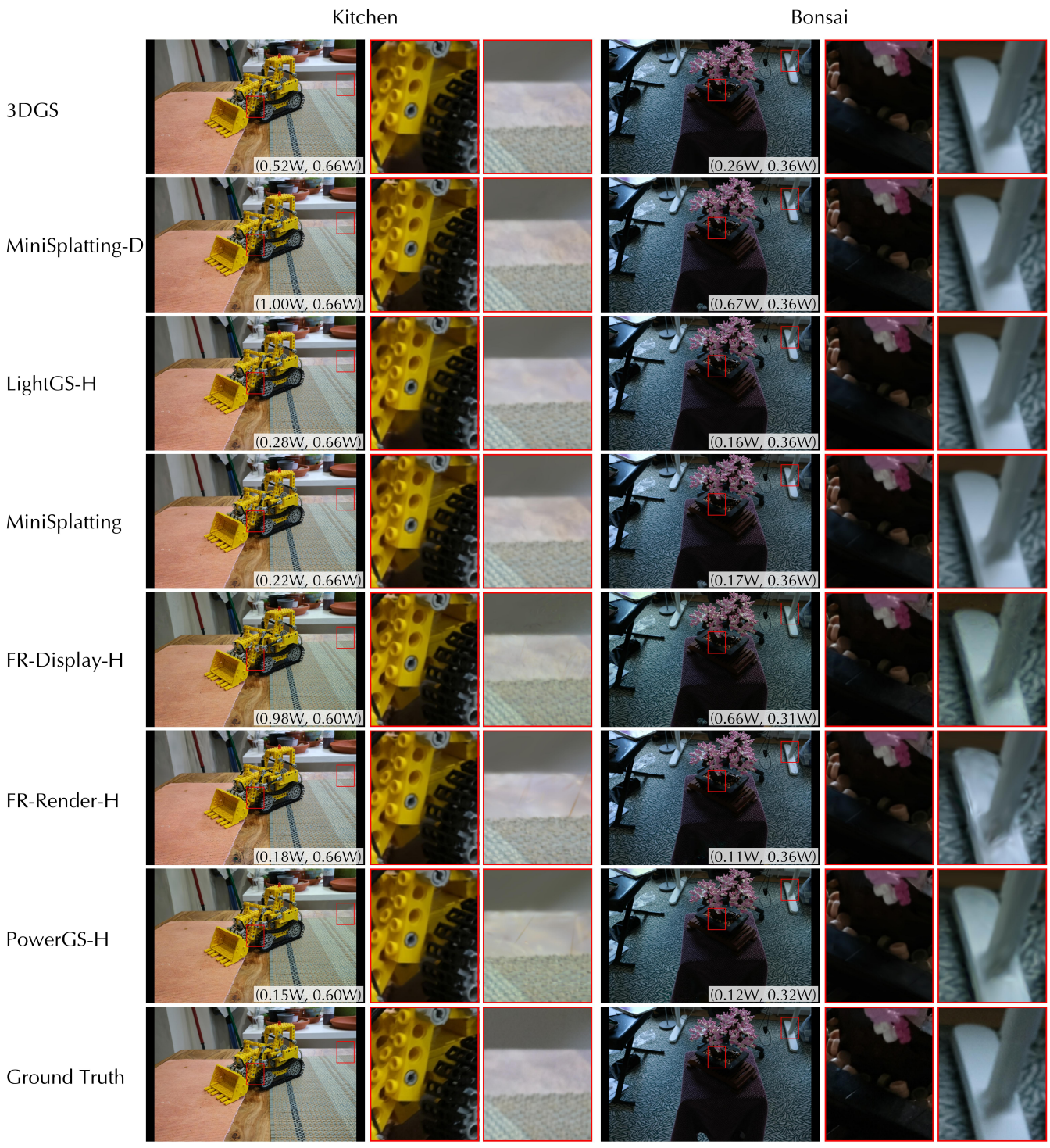

Fig. 5. **Qualitative Evaluation.** Rendered images of two scenes in Mip-NeRF360 are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

Fig. 6. **Qualitative Evaluation.** Rendered images of two scenes in Mip-NeRF360 are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

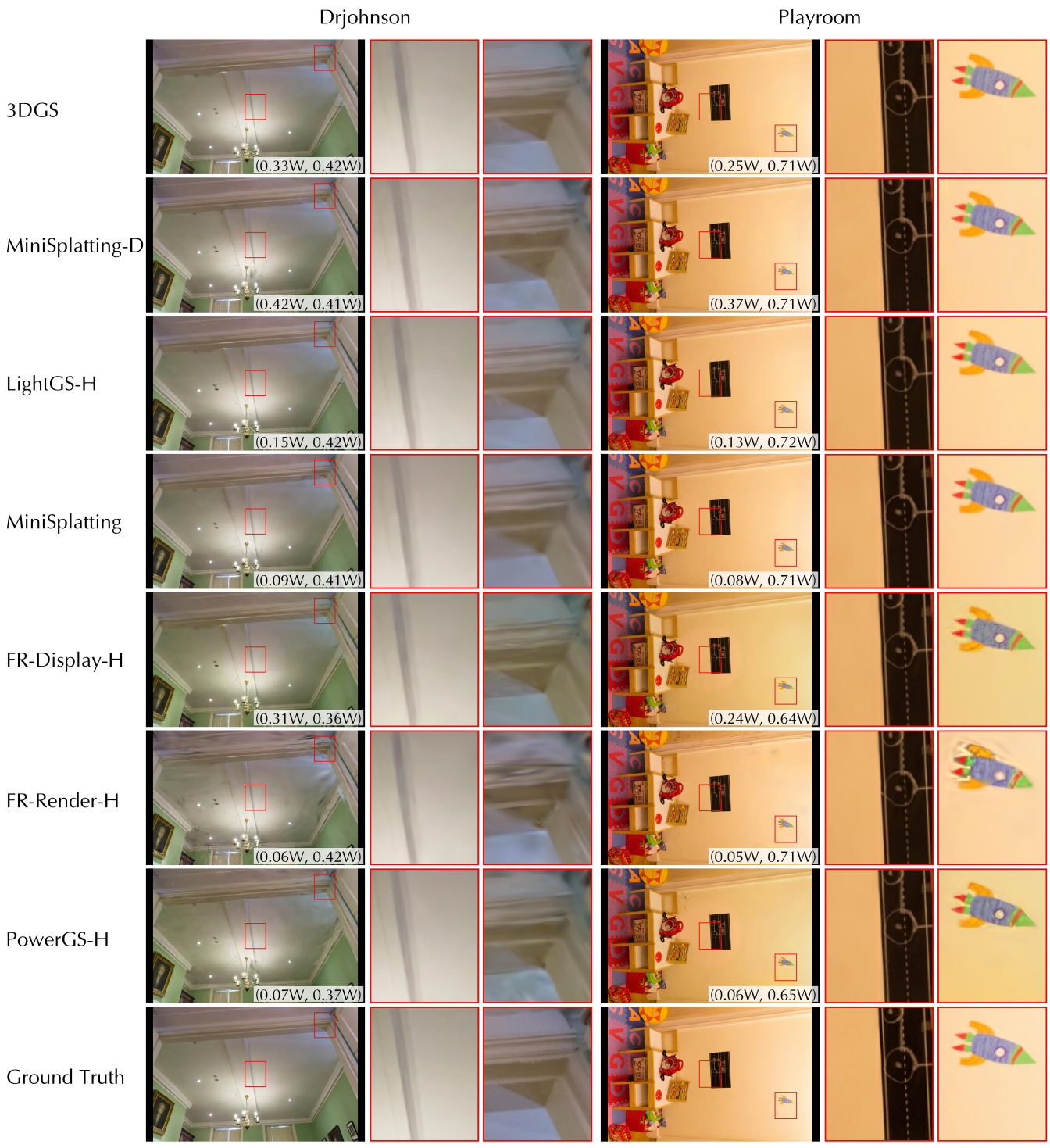

Fig. 7. **Qualitative Evaluation.** Rendered images of two scenes in Deep Blending are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

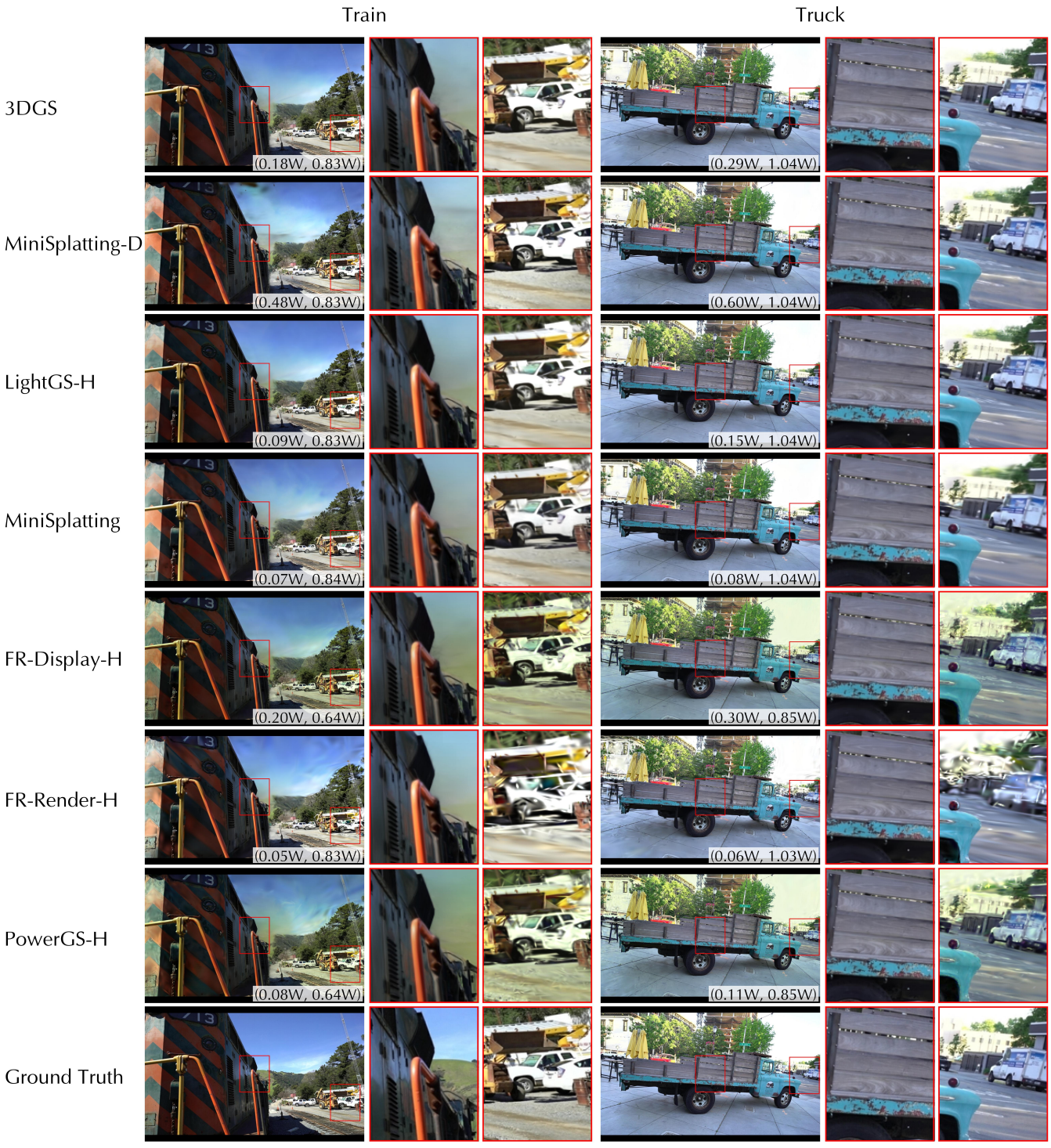

Fig. 8. **Qualitative Evaluation.** Rendered images of two scenes in Tank&Temple are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

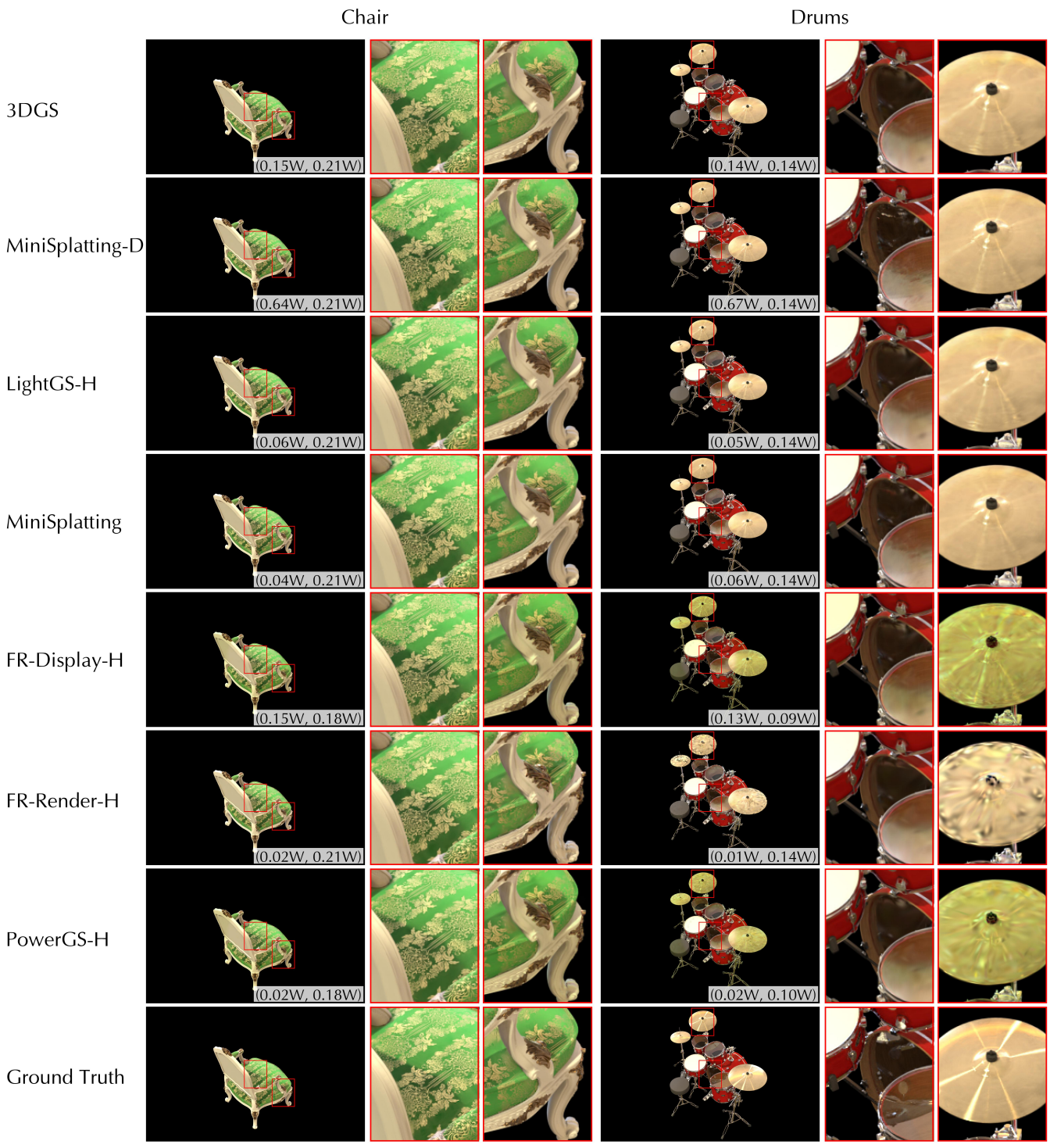

Fig. 9. **Qualitative Evaluation.** Rendered images of two scenes in Synthetic NeRF are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

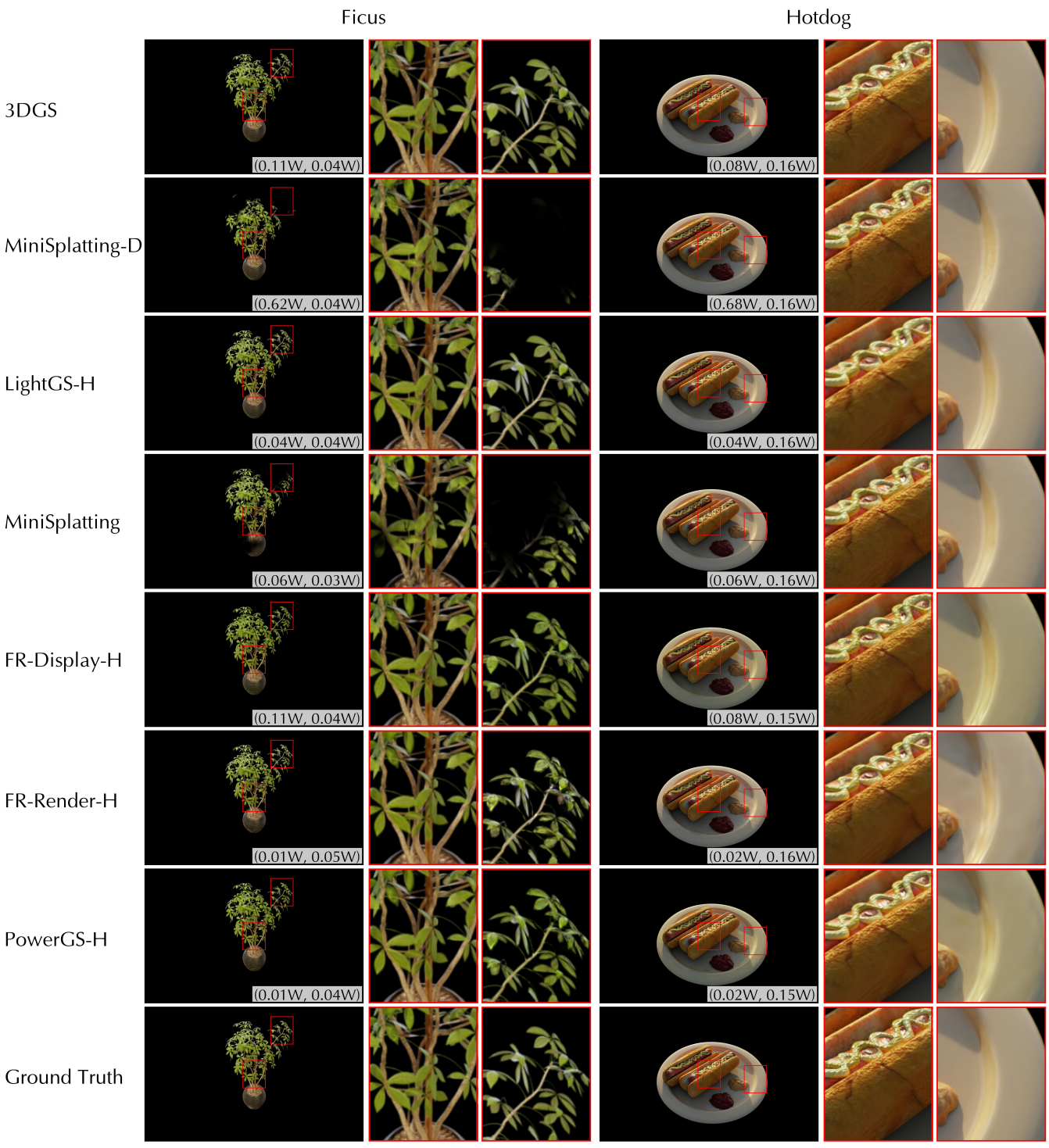

Fig. 10. **Qualitative Evaluation.** Rendered images of two scenes in Synthetic NeRF are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

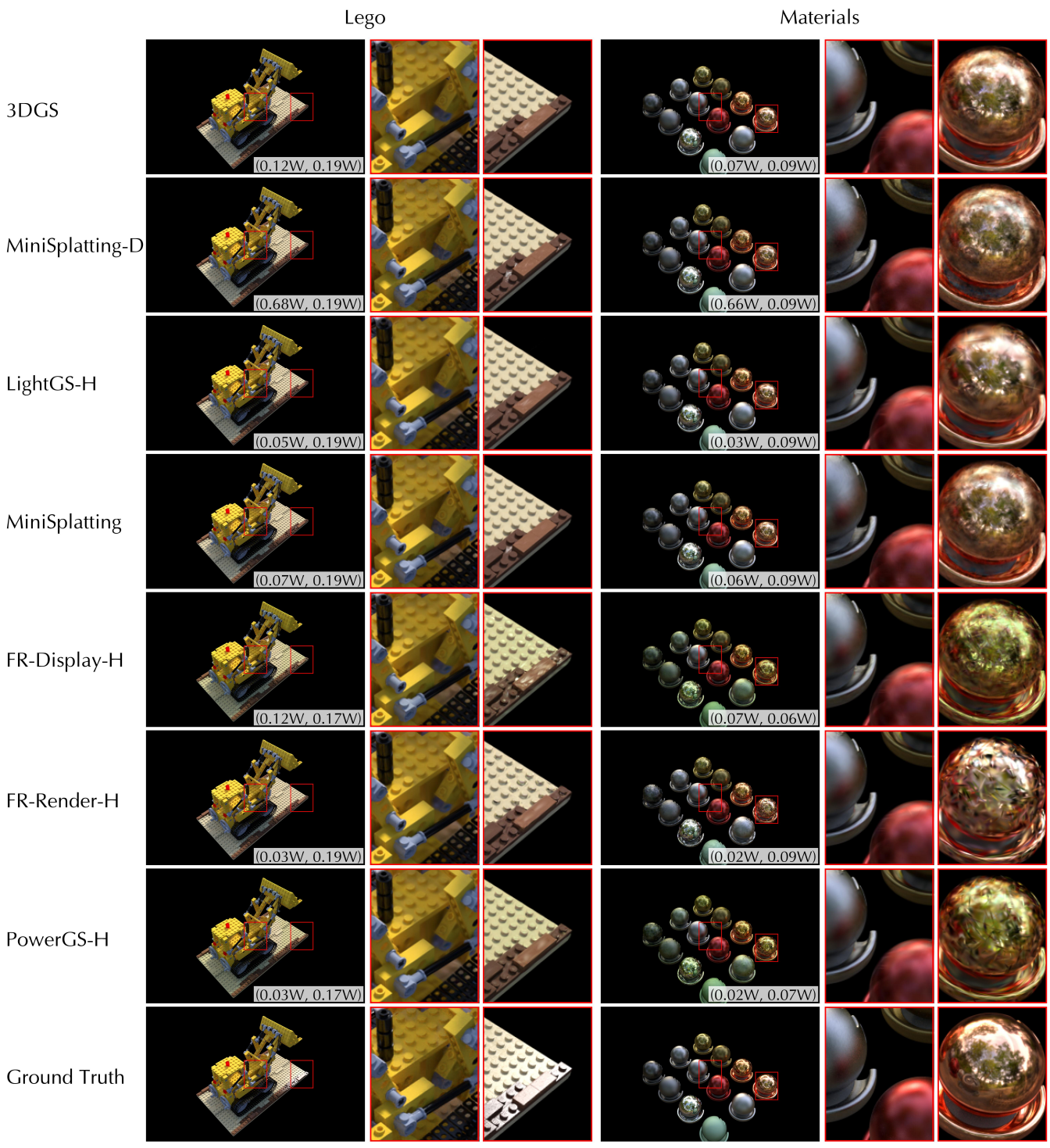

Fig. 11. **Qualitative Evaluation.** Rendered images of two scenes in Synthetic NeRF are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.

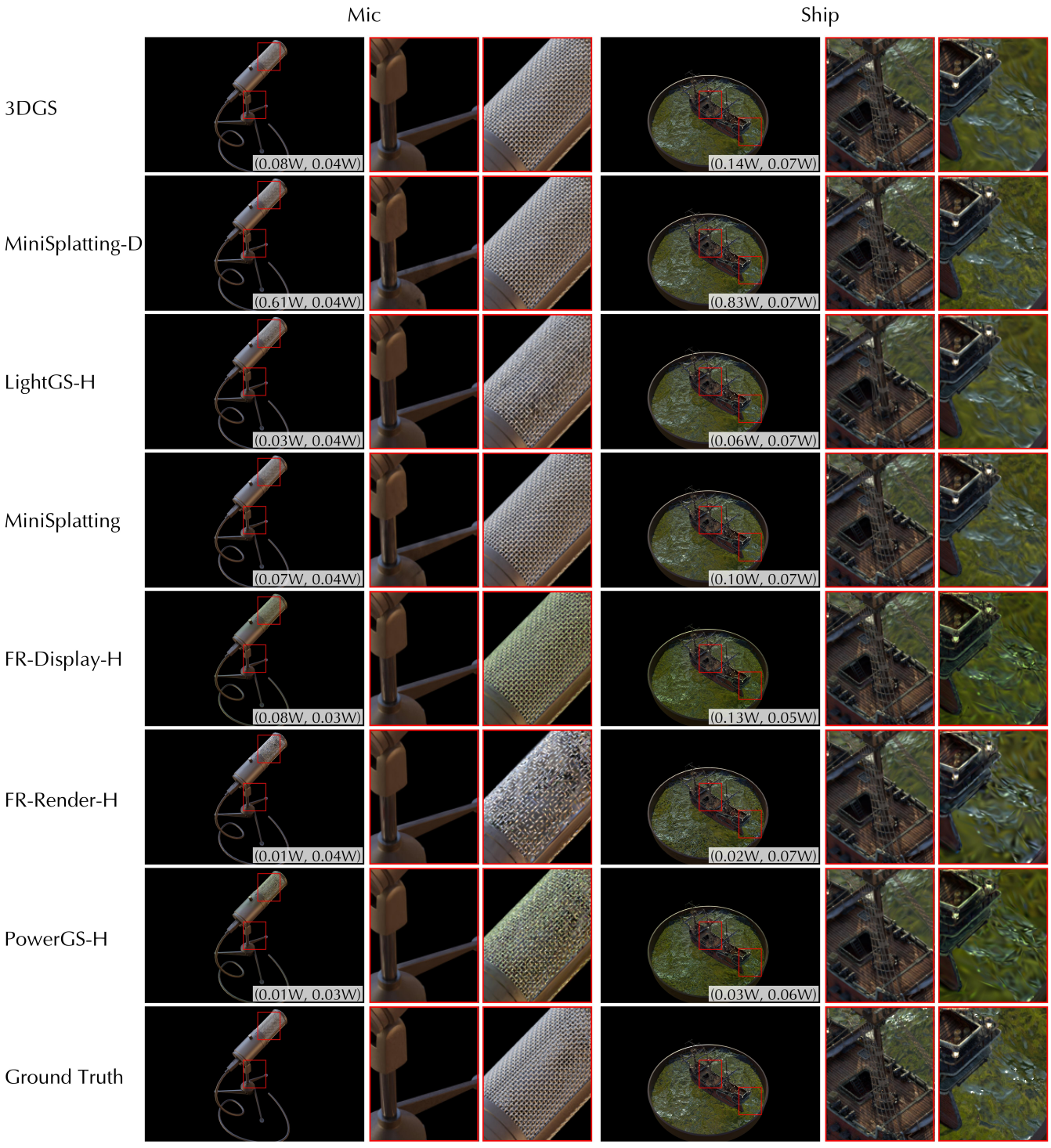

Fig. 12. **Qualitative Evaluation.** Rendered images of two scenes in Synthetic NeRF are shown, with annotated values for (rendering power, display power). Zoom-in views of the fovea region (left) and the periphery (right) are also provided. For all foveated versions, the gaze is fixed at the center of the image.