RainyGS: Efficient Rain Synthesis with Physically-Based Gaussian Splatting (Supplementary Material)

QIYU DAI*, School of Intelligence Science and Technology, Peking University, China
XINGYU NI*, School of Computer Science, Peking University, China
QIANFAN SHEN, School of EECS, Peking University, China
WENZHENG CHEN†, Wangxuan Institute of Computer Technology, Peking University, China
BAOQUAN CHEN†, School of Intelligence Science and Technology, Peking University, China
MENGYU CHU†, School of Intelligence Science and Technology, Peking University, China

The supplementary materials offer a detailed explanation of our RainyGS, covering Scene Modeling in Sec. A, Auxiliary Map Extraction in Sec. B, Rain Simulation on Height Maps in Sec. C, and Screen-Space Reflection in Sec. D. Additionally, we present results and performance analyses from various experimental scenes in Sec. E. To further illustrate our method, we include a video demonstrating the dynamic synthesis of rain.

A DETAILS OF SCENE MODELING

A.1 Normal Prior Supervision

We employ PGSR [Chen et al. 2024] as a unified module for appearance and geometry reconstruction. While PGSR achieves state-of-the-art quality, it still exhibits noticeable artifacts in texture-less regions and areas with transparent or reflective materials, such as floors, walls, and car windows (as shown in Fig. 9). We demonstrate that introducing normal priors from a pretrained monocular normal estimation model [Bae and Davison 2024] to supervise the rendered normal maps can effectively improve both the rendering fidelity and geometric accuracy of PGSR. This further leads to higher-quality auxiliary maps, including depth maps, normal maps, and height maps, which are essential for downstream rain simulation.

A.2 Comparison of Scene Modeling Methods

An alternative approach to scene modeling is combining a decomposed Radiance Field (e.g., GaussianShader [Jiang et al. 2024]) with a geometry reconstruction method (e.g., GOF [Yu et al. 2024]). Specifically, GaussianShader serves as the appearance module to disentangle appearance and illumination, while GOF is employed as the geometry module to extract detailed scene structures. To maintain consistency between these two modules, multi-view depth maps generated by the GOF model are used to supervise the training of GaussianShader. The advantage of this approach lies in its Physically-Based Rendering (PBR) formulation, enabling more accurate light transport and relighting. However, this method requires training two separate models, resulting in redundancy and increased computational cost. Additionally, the high degree of flexibility in GaussianShader makes optimization challenging, often leading to suboptimal visual quality.

In contrast, PGSR adopts a unified model that achieves superior appearance and geometry quality, while remaining efficient during both training and inference. Although this approach sacrifices PBR properties in scene modeling, environment maps can still be replaced in the Water Rasterization stage. As illustrated in Fig. 10, our method employing PGSR achieves comparable lighting effects to the GShader+GOF pipeline, while delivering higher rendering quality.

*joint first authors
†corresponding authors

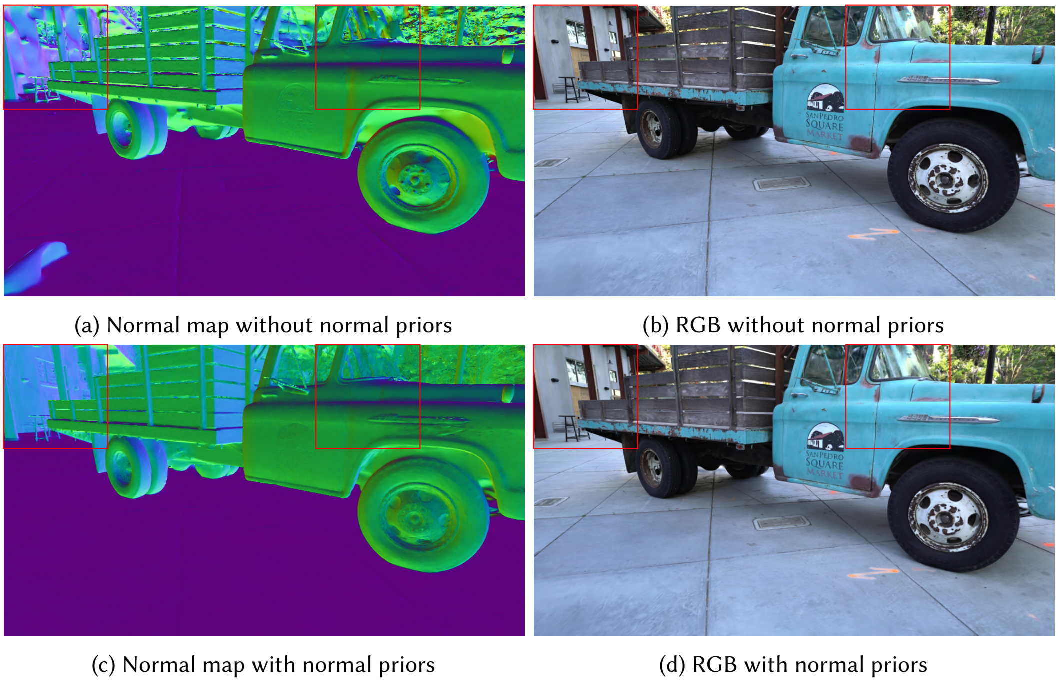

(a) Normal map without normal priors
(b) RGB without normal priors
(c) Normal map with normal priors
(d) RGB with normal priors

Fig. 9. As highlighted in the red box, leveraging priors from a pretrained monocular normal estimation model leads to notable improvements in both rendering fidelity and geometric precision of PGSR for scene modeling.

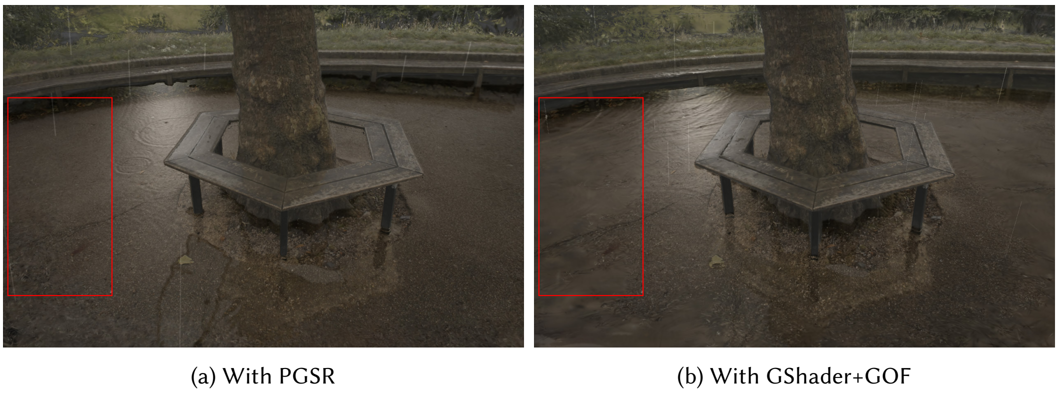


Fig. 10. Compared to the GShader+GOF pipeline (b), using PGSR (a) achieves higher rendering quality while maintaining comparable lighting and shadow effects.

B DETAILS OF AUXILIARY MAP PREPARATION

We model rain as spherical 3D Gaussians for auxiliary map extraction. For water surface Gaussians, their positions are aligned with the 3D coordinates corresponding to each pixel in the height map, with a fixed radius of 0.006. The normal of each Gaussian is computed from the normal map derived from the height map. This enables the extraction of normal maps for specific viewpoints using 3DGS rasterization. We further model the rain streaks as densely aligned lines of 3D Gaussian spheres, each with a radius of 0.006 and a near-white RGB color of [200, 200, 200].

C DETAILS OF RAIN SIMULATION ON HEIGHT MAPS

Our simulation framework begins with the two-dimensional staggered MAC grid [Harlow and Welch 1965] as usual, storing components of the velocity $\boldsymbol{u}$ at the appropriate edge midpoints and the depth h at the cell centers. Based on the philosophy of time splitting, the simulation pipeline within a time step Δt are summarized in the following text.

(1) **Handling Advection.** The velocity field $\boldsymbol{u}$ and the height field h are treated with the semi-Lagrangian method and the 1st-order upwind scheme, respectively:

$$\boldsymbol{u}^* \leftarrow \text{SemiLagrangian}(\boldsymbol{u}^n, \Delta t, \boldsymbol{u}^n), \tag{13}$$

$$h^* \leftarrow \text{FirstOrderUpwind}(\boldsymbol{u}^n, \Delta t, h^n). \tag{14}$$

(2) **Enhancing Stability.** Height values that are overly close to 0 are clamped to avoid undershooting:

$$h^* \leftarrow 0, \qquad \text{if } h^* < 10^{-6}. \tag{15}$$

(3) **Constructing heights.** The total height field η is calculated by

$$\eta \leftarrow H + h^*. \tag{16}$$

(4) **Maintaining Rains.** Existing raindrops evolve, and new raindrops are generated.

(5) **Applying pressure.** The influence of pressure force is taken into account:

$$\boldsymbol{u}^* \leftarrow \boldsymbol{u}^* - \Delta t\, g\, \boldsymbol{\nabla}\eta, \tag{17}$$

in which g denotes the gravitational acceleration.

(6) **Extrapolating Velocities.** It is important to extrapolate velocities from wet regions into dry regions, in order to handle boundary conditions.

D DETAILS OF SCREEN-SPACE REFLECTION

Screen-Space Reflection (SSR) [McGuire and Mara 2014] is a real-time rendering technique that approximates reflective surfaces by reusing the existing information available in the screen space, such as the depth map and rendered color (RGB) map. The SSR avoids the computational complexity of full ray-tracing by confining reflection computation to the visible scene, whose process can be divided into several stages:

(1) **Ray Casting.** The algorithm begins by casting rays from the view position of each reflective pixel. The ray direction is determined by the surface normal at the pixel and the direction to the camera, adhering to the law of reflection:

$$\boldsymbol{R} = 2(\boldsymbol{N} \cdot \boldsymbol{V})\boldsymbol{N} - \boldsymbol{V}, \tag{18}$$

where $\boldsymbol{R}$ is the reflection direction, $\boldsymbol{N}$ is the surface normal. and $\boldsymbol{V}$ is the view vector.

(2) **Ray Marching.** Once the reflection direction is determined, it will be projected onto the screen space to form a 2D directional vector $\boldsymbol{d}$. Then, ray marching is performed in screen space from the reflective pixel along this direction. We use the digital differential analyzer (DDA) algorithm to iteratively sample the projected ray.

(3) **Recovering Points.** It is important to note that values save in the depth map are actually the z component of positions in the camera coordinate system. Every time we sample a point from the projected ray, its corresponding point on the original ray should be recovered. We use D to denote the value saved in the depth map at this pixel and use Z to denote the z component of the corresponding ray point in camera coordinates.

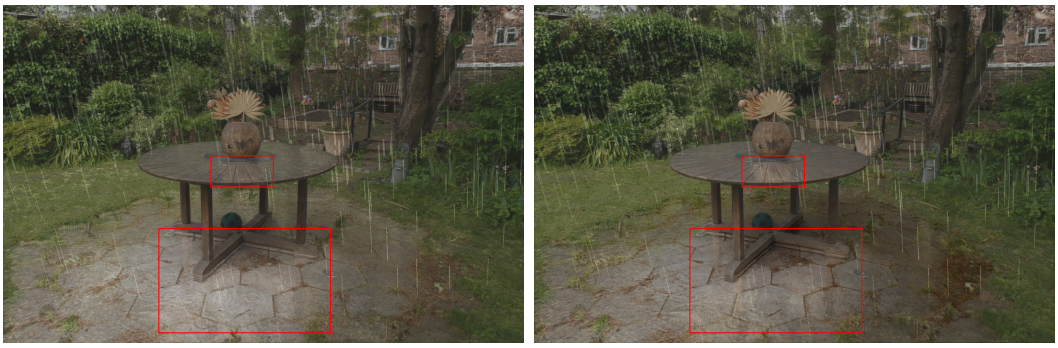

(a) With Ray-Tracing Reflection
(b) With Screen-Space Reflection

Fig. 11. **Garden Scenario.** Compared to ray-traced reflection rendering (a), SSR-based reflection rendering (b) achieves nearly identical visual quality while offering significantly higher efficiency and reduced memory consumption. The reflection regions are highlighted with red boxes.

(4) **Collision Detection.** The ray is assumed to collide objects if and only if the following condition met:

$$D \leq Z < D + \varepsilon, \tag{19}$$

in which the parameter ε is used to determine the surface thickness. For most of our test scenarios, we find that $\varepsilon = 3 \times 10^{-2}$ is a good setting.

(5) **Fetching Reflective Colors.** When a collision is detected, the color saved in the pixel where the collision occurs is fetched. Consequently, the process of ray marching terminates. Otherwise, if no collision is detected and the projected ray is outside the screen space, the program also exists and sampled the color from the environment map using direction $\boldsymbol{R}$ as a fallback.

Discussion To evaluate the performance of ray tracing (RT) versus screen space reflection (SSR) in reflection computation, we adopt the approach described in [Gao et al. 2024]. Specifically, we trace the reflected light paths of water surface Gaussians, identify intersections with original scene Gaussians, and apply alpha blending to compute their reflection colors. These results were then rasterized to generate a reflection map (I_{highl} in the main text), which is integrated into our rendering pipeline. The final output combines this reflection map with the original image and refraction, as depicted in Fig. 11 (a).

As illustrated in (b), our SSR implementation achieves a reflection quality comparable to RT. Moreover, as shown in Table 1, our method delivers a 60× speedup and a 1.65× reduction in peak memory usage. These performance gains are attributed to the computational efficiency of SSR, which approximates reflections by stepping through a limited number of screen-space pixels. In contrast, the RT approach requires time-consuming traversal of a BVH tree constructed from scene Gaussians and reordering of Gaussians along the light path.

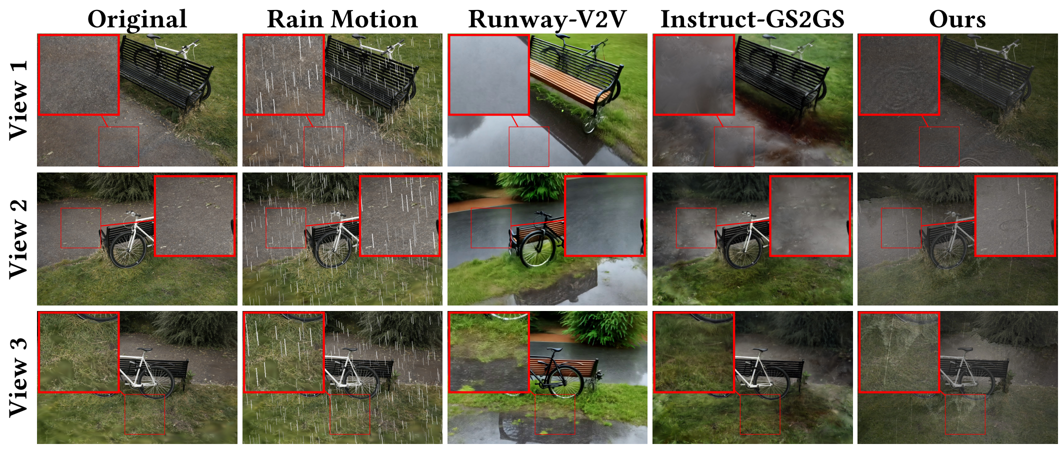


Fig. 12. **Bicycle Scenario.** Rain synthesis results from different viewpoints at the same timestep. Runway-V2V struggles to preserve 3D consistency, while Rain Motion and Instruct-GS2GS fail to generate realistic rain streaks, puddles, and ripples. In contrast, RainyGS maintains 3D consistency and produces realistic rain streaks and water accumulation.

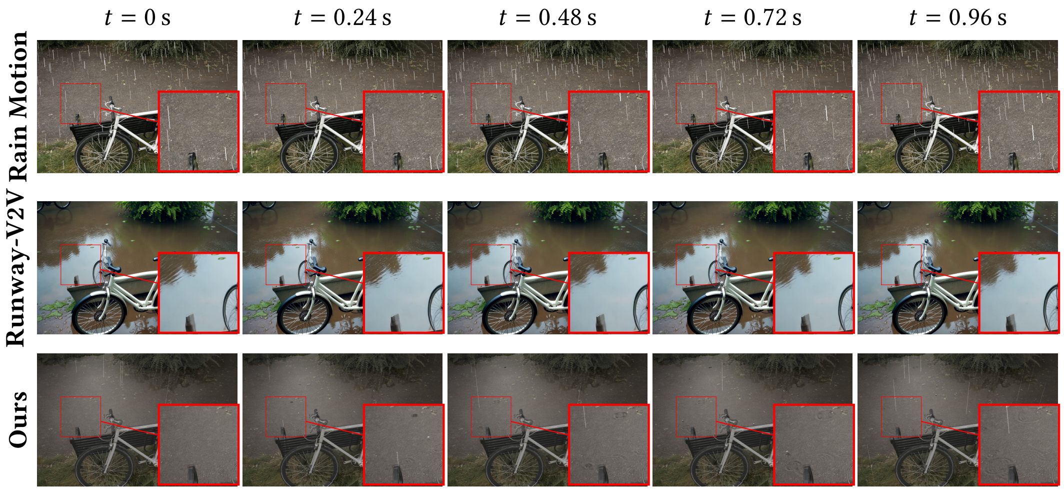


Fig. 13. **Bicycle Scenario.** Rain synthesis results from the same viewpoint at different timesteps are presented. Rain Motion generates unrealistic, random rain droplets without hydrops, while Runway-V2V generates physically inaccurate and unrealistic ripples. In contrast, RainyGS creates realistic, time-evolving hydrops and puddles.

E MORE RESULTS

E.1 More Results of Comparison with Video-Based Rain Synthesis

We evaluate our method on the Garden, Treehill, and Bicycle scenes from the MipNeRF360 dataset, as well as the Family and Truck scenes from the Tanks and Temples dataset. For comparison,

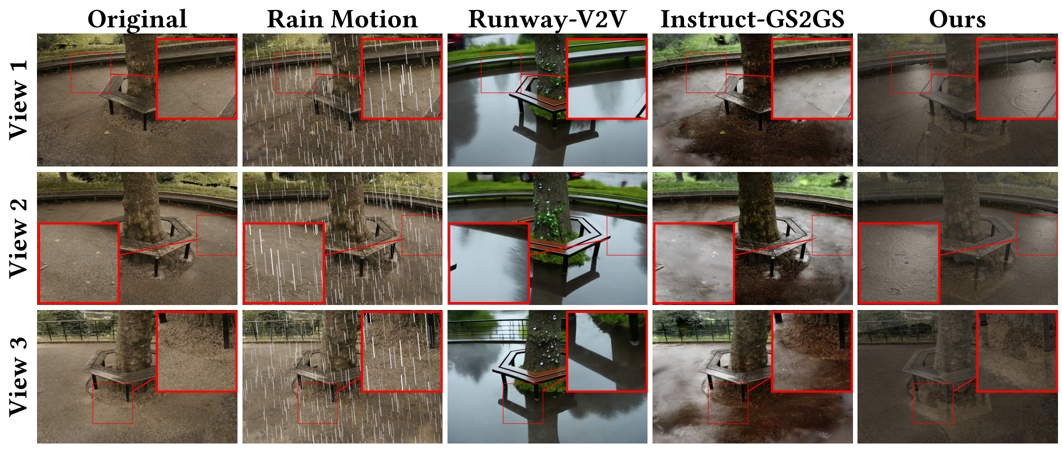


Fig. 14. **Treehill Scenario.** Comparison of rain synthesis at the same timestep from various perspectives. Runway-V2V struggles with 3D consistency, and Rain Motion and Instruct-GS2GS cannot produce realistic rain streaks, puddles, or ripples. By contrast, RainyGS achieves stable 3D consistency and faithfully renders rain streaks and water accumulation.

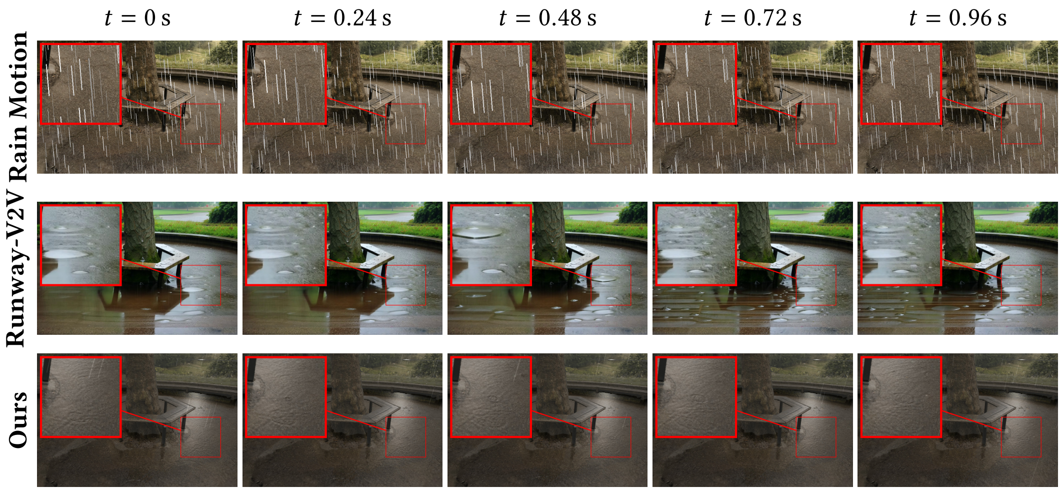


Fig. 15. **Treehill Scenario.** Results of rain synthesis at different moments in time from the same viewpoint are displayed. Rain Motion fails to produce realistic rain, creating random droplets without hydrops, and Runway-V2V outputs almost static rain. RainyGS generates realistic and dynamic hydrops and puddles.

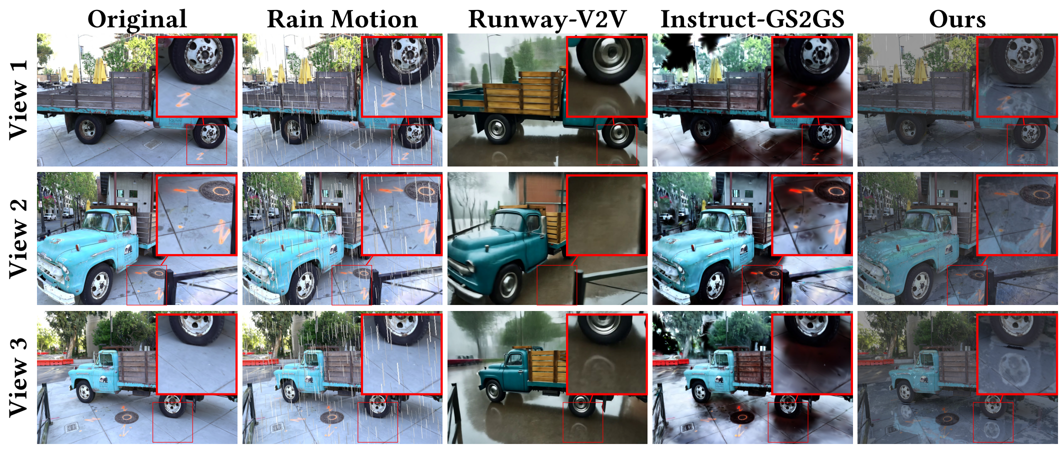


Fig. 16. **Truck Scenario.** Results of rain synthesis at an identical timestep observed from multiple views. Runway-V2V fails to maintain 3D consistency, and both Rain Motion and Instruct-GS2GS lack realistic rain streaks, puddles, and ripples. In contrast, RainyGS maintains 3D consistency and generates lifelike rain streaks and puddles.

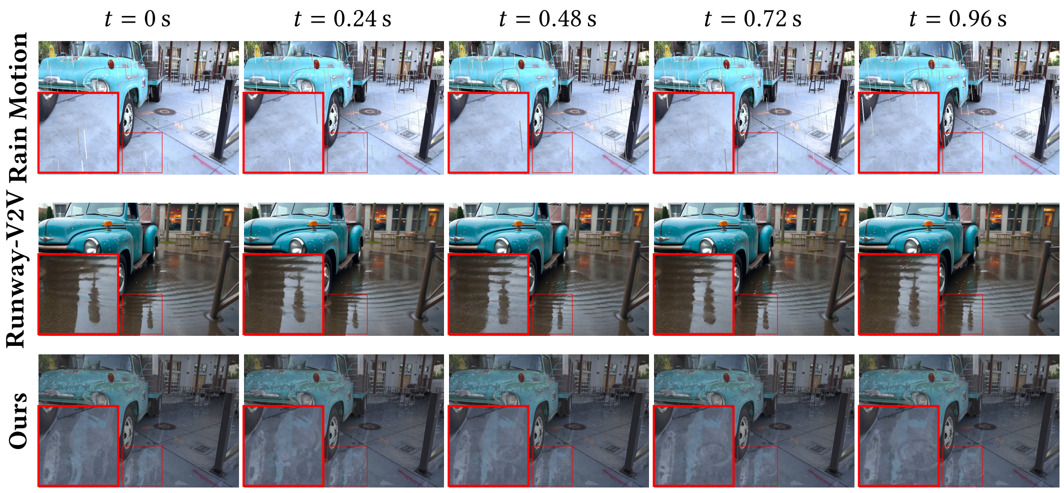


Fig. 17. **Truck Scenario.** Rain synthesis outputs from the same perspective over different timesteps are illustrated. Rain Motion creates artificial and random droplets with no hydrops, and Runway-V2V produces ripples that are neither physically plausible nor visually realistic. RainyGS, on the other hand, generates dynamic hydrops and puddles with temporal realism.

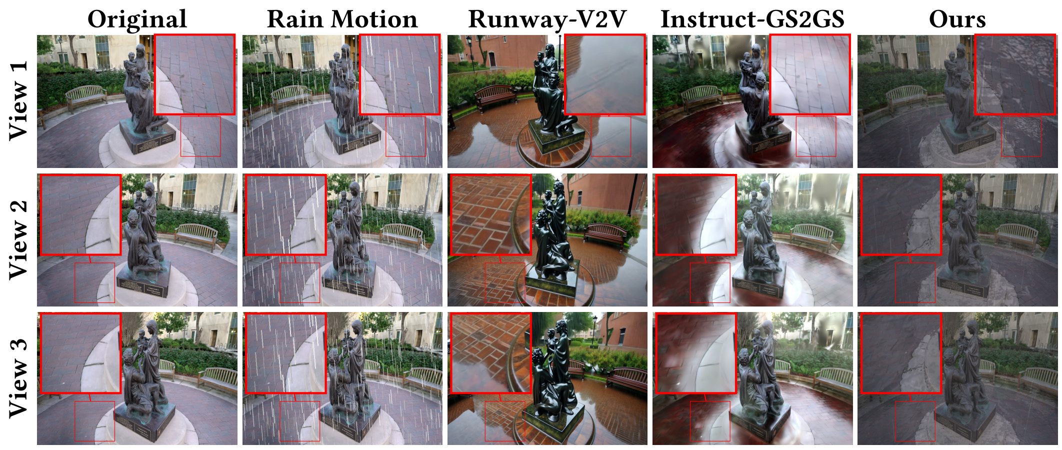


Fig. 18. **Family Scenario.** Rain synthesis outputs from several viewpoints at the same moment in time. Runway-V2V struggles to preserve 3D consistency, while Rain Motion and Instruct-GS2GS are unable to generate realistic rain streaks, puddles, and ripples. In contrast, RainyGS preserves 3D consistency and produces visually convincing rain streaks and water accumulation.

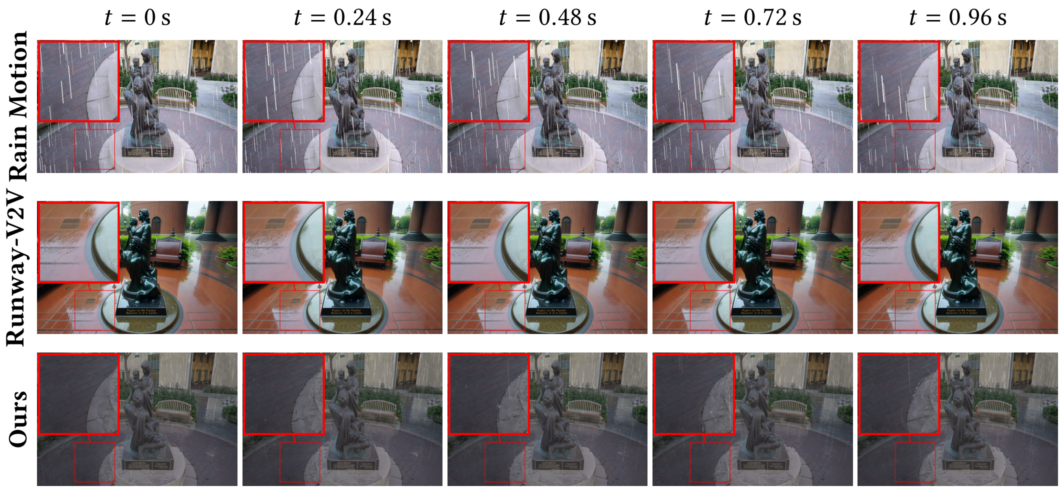
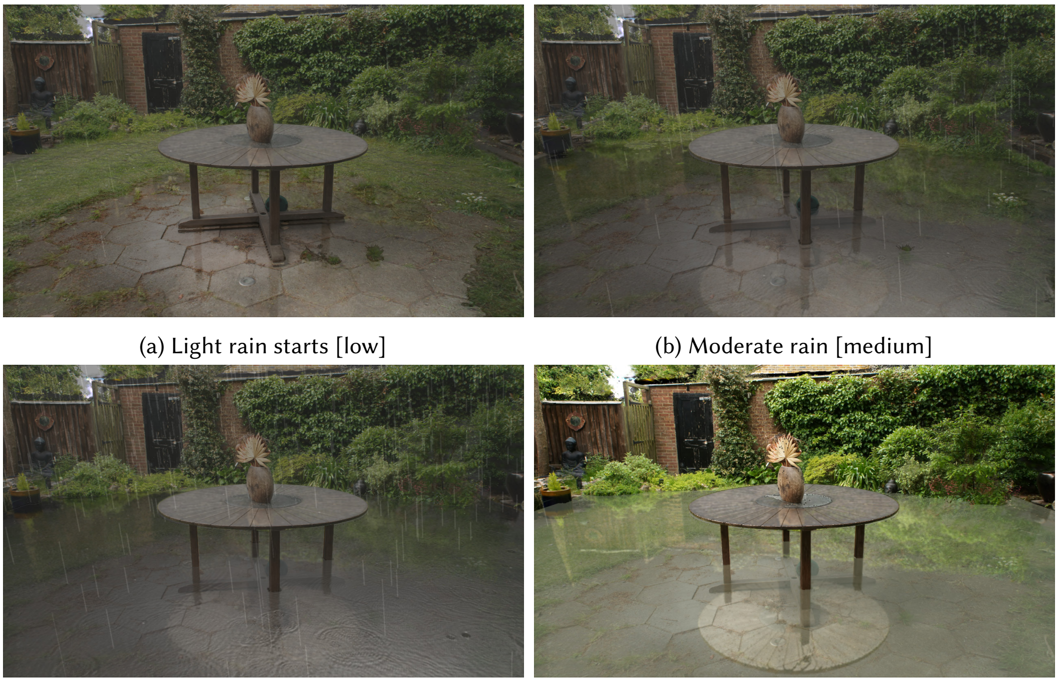


Fig. 19. **Family Scenario.** Shown are rain synthesis results at different timesteps from the same viewpoint. Rain Motion produces random, unrealistic rain droplets and lacks hydrops, while Runway-V2V yields ripples that lack physical correctness and visual authenticity. RainyGS, however, simulates realistic hydrops and puddles that evolve over time.

we used two baselines: Rain Motion [Wang et al. 2022], a video rain synthesis method; Runway-V2V [Runway 2024a,b], a state-of-the-art commercial video-to-video generation model; and Instruct-GS2GS [Vachha and Haque 2024], a recent text-driven 3DGS editing method. For Runway-V2V, we use the prompt: *Raining in the scene, with water puddles on the ground, reflections of scenes and lights on wet pavement, and raindrops creating ripples on the surface, under an overcast sky.* In addition to the Garden scene presented in Fig. 5 and Fig. 6 of the main text, we further show the Bicycle, Treehill, Truck, and Family scenes in Fig. 12 - Fig. 19.

The results generated by Runway-V2V demonstrate a general rainy visual effect. Nevertheless, its output exhibits several shortcomings. First, the appearance and geometry of scene objects are often altered; for example, in Fig. 13, the bench changes from dark brown to light brown, and the bicycle shifts from white to black. Second, the generated videos suffer from oversaturated colors. In addition, Runway-V2V lacks physical accuracy, resulting in artifacts such as unrealistic ripples (e.g., Fig. 15) and incorrect reflections (e.g., the wheel in Fig. 16). By contrast, Rain Motion is limited to applying simple 2D rain effects on video frames, without 3D multiview consistency or style adaptation. This leads to inferior visual fidelity and poor physical realism. Instruct-GS2GS also falls short, as it lacks advanced rain phenomena such as water accumulation, rain streaks, and dynamic reflections, and is restricted to static edits.

In contrast, our method consistently outperforms all baselines by generating highly realistic rain effects while preserving physical plausibility, including fluid dynamics (e.g., Fig. 12), accurate reflections (e.g., Fig. 16), and refractions (e.g., Fig. 18). Moreover, it maintains multi-view consistency and ensures temporally coherent dynamics.



(a) Light rain starts [low]

(b) Moderate rain [medium]

(c) Heavy rain [high]

(d) Rain ends [high]

Fig. 20. Rain progression from start to end, showing the control of rain intensity, water volume (indicates with []), and lighting.

Table 2. User Study Results on Amazon Mechanical Turk, reporting the percentage (%) of cases where users preferred our method over each baseline for both image and video comparisons.

	Image	**Video**
vs Rain Motion	70.9	67.5
vs Runway-V2V	70.7	72.0
vs Instruct-GS2GS	65.2	65.7

Table 3. The computational statistics for different scenes. The time here is measured per frame averagely. Here, the simulation time includes that cost for data transferring from height maps to Gaussians.

Scenes	**# Gaussians**	**Precomputed Time**	**Rendering Time**	**Peak Video Memory**
Garden	4.290 M	0.013 s	0.032 s	8.561 GB
Truck	4.368 M	0.022 s	0.035 s	9.044 GB
Treehill	8.169 M	0.049 s	0.039 s	18.049 GB
Family	8.524 M	0.050 s	0.039 s	19.166 GB
Bicycle	10.687 M	0.053 s	0.041 s	23.511 GB

E.2 User Study

Tab. 2 summarizes the results of a user study conducted on Amazon Mechanical Turk, designed to evaluate the perceptual quality of our method compared to existing baselines. A total of 54 participants were recruited, each asked to judge 30 randomly selected examples consisting of 15 images and 15 videos. For each example, users indicated their preferred result in a pairwise comparison setting. The results show a clear and consistent preference for our method across both image and video modalities, highlighting its effectiveness and perceptual superiority over competing approaches.

E.3 Additional Results of Precise User Control

RainyGS supports precise, physics-based user control over key parameters, including rain intensity, water level, and scene brightness, allowing flexible adjustment to meet various simulation needs. As illustrated in Fig. 20, we present the rain progression over time on MipNeRF360-Garden, showcasing dynamic variations in rain intensity, gradual water accumulation, and changing lighting conditions, all rendered consistently from the same scene and camera viewpoint.

E.4 More Results and Analysis of Performance

Our experiments are performed on a platform with a Nvidia Geforce RTX 3090 Graphics Card, whose memory capacity is 24 GB. Here we report the computational statistics for the results given in the main text, as well as those provided in Fig.12 - Fig. 19, shown in Tab. 3. Generally speaking, our method, namely RainyGS, takes the most of time in simulation and transferring data, which can be pre-computed. As to the rendering stage, the speed is around 30 FPS. The memory usage is roughly linear to the number of Gaussians, which fits the capacity of personal computers.

E.5 Additional Results of Downstream Tasks

As shown in Fig. 21, RainyGS can be utilized for downstream tasks in autonomous driving. For instance, it efficiently transforms continuous driving scenarios, such as Waymo scenes, into rainy weather conditions. This capability facilitates robustness testing of detection algorithms and provides a new method for simulating adverse weather and generating synthetic data for autonomous driving research.

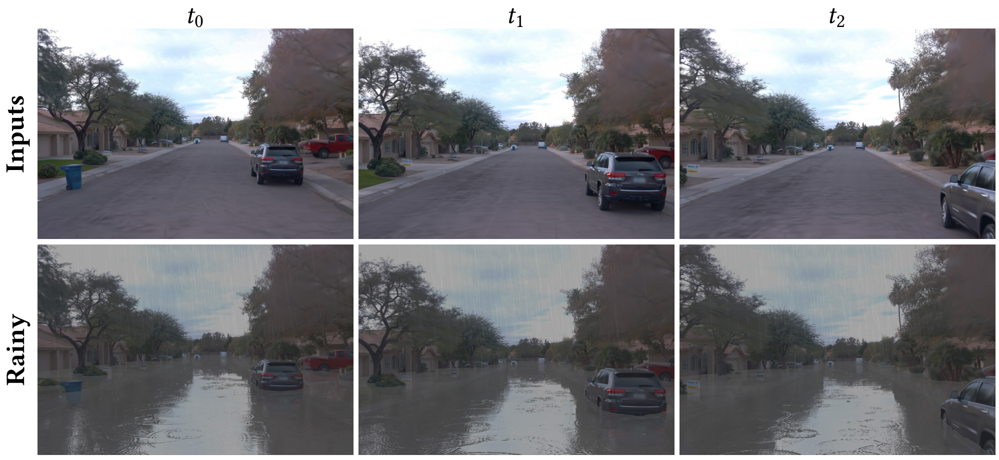


Fig. 21. Downstream Application: Rainy Weather Synthesis for Waymo Scenes. Our method facilitates the efficient transformation of continuous autonomous driving scenes into rainy weather conditions, enabling robustness testing of autonomous driving perception algorithms. Additionally, it offers a novel framework for synthesizing adverse weather data in autonomous driving.

REFERENCES

Gwangbin Bae and Andrew J Davison. 2024. Rethinking inductive biases for surface normal estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9535–9545.

Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. 2024. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* (2024).

Jian Gao, Chun Gu, Youtian Lin, Zhihao Li, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. 2024. Relightable 3D Gaussians: Realistic Point Cloud Relighting with BRDF Decomposition and Ray Tracing. arXiv:2311.16043 [cs.CV]

Francis H. Harlow and J. Eddie Welch. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids* 8, 12 (1965), 2182–2189.

Yingwenqi Jiang, Jiadong Tu, Yuan Liu, Xifeng Gao, Xiaoxiao Long, Wenping Wang, and Yuexin Ma. 2024. Gaussianshader: 3d gaussian splatting with shading functions for reflective surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5322–5332.

Morgan McGuire and Michael Mara. 2014. Efficient GPU Screen-Space Ray Tracing. *Journal of Computer Graphics Techniques (JCGT)* 3, 4 (9 December 2014), 73–85.

Runway. 2024a. Gen-3 Alpha Video to Video. https://academy.runwayml.com/gen3-alpha/gen3-alpha-video-to-video.

Runway. 2024b. Introducing Gen-3 Alpha: A New Frontier for Video Generation. https://runwayml.com/research/introducing-gen-3-alpha.

Cyrus Vachha and Ayaan Haque. 2024. Instruct-GS2GS: Editing 3D Gaussian Splats with Instructions. https://instruct-gs2gs.github.io/

Shuai Wang, Lei Zhu, Huazhu Fu, Jing Qin, Carola-Bibiane Schönlieb, Wei Feng, and Song Wang. 2022. Rethinking video rain streak removal: A new synthesis model and a deraining network with video rain prior. In *European Conference on Computer Vision*. Springer, 565–582.

Zehao Yu, Torsten Sattler, and Andreas Geiger. 2024. Gaussian Opacity Fields: Efficient Adaptive Surface Reconstruction in Unbounded Scenes. *ACM Transactions on Graphics* (2024).