Generative World Renderer at the Speed of Play

Guixu Lin[1], **Zheng-Hui Huang**[1], **Siqi Yang**[1], **Ming-Hsuan Yang**[2], **Kaipeng Zhang**[1], **Zhixiang Wang**[1,*]

[1]Alaya Lab [2]University of California, Merced

Generative world renderer **AlayaRenderer** receives structured world states exported from physics engines and synthesizes RGB frames. Unlike models that generate frames from text/control-hints prompts, AlayaRenderer preserves scene structure without altering the underlying world dynamics. This demonstrates an alternative path toward interactive world modeling and user-controllable play. However, the original AlayaRenderer is too computationally expensive for real-time deployment. This technical report introduces **AlayaRenderer-Flash**, a real-time-oriented generative forward world renderer that pushes AlayaRenderer from **0.56 FPS** to **31.54 FPS**, reaching the speed of play. AlayaRenderer-Flash reformulates the original renderer as a few-step autoregressive streaming model and introduces lightweight distilled codecs for efficient latent encoding and frame reconstruction. It retains the teacher model's G-buffer and text-prompt interfaces while enabling continuous rendering over input streams of unbounded length. We evaluate AlayaRenderer-Flash on G-buffer streams across content preservation, temporal consistency, cross-window stability, prompt controllability, and runtime efficiency. Our results show that AlayaRenderer-Flash substantially reduces inference cost while preserving the core rendering capabilities of the teacher model. By integrating AlayaRenderer-Flash with a physics engine, we build a fully playable generative world running at 30 FPS.

Demo: https://alaya-renderer-flash.alayalab.ai
Correspondence: Zhixiang Wang <zhixiang.wang@shanda.com>
Date: July 22, 2026

ALAYA LAB

Figure 1 AlayaRenderer-Flash restyling a live *SuperTuxKart* session in our interactive web demo. The game engine continuously exports synchronized G-buffers to AlayaRenderer-Flash, which generates stylized RGB frames at playback rate. Players can modify the visual style of the rendered game through arbitrary text prompts while the underlying gameplay logic remains unchanged.

1 Introduction

Generative models offer a promising path toward interactive visual worlds, but methods that synthesize frames solely from text prompts or control hints often struggle to preserve scene structure, material properties, and gameplay logic as the world evolves. A more principled alternative is to keep the physics engine responsible for simulating the world while delegating only the appearance generation to a generative renderer. Fortunately, modern game engines already expose a natural interface for this paradigm. In deferred rendering, scenes are first rasterized into physical and geometric buffers—including depth, normals, albedo, roughness, and metallicity—before being shaded into RGB frames (Saito and Takahashi, 1990; Nalbach et al., 2017; Thies et al., 2019). These G-buffers encode the structured world state while leaving the final appearance unconstrained. Recent diffusion- and flow-based renderers have demonstrated that generative models conditioned on G-buffers can replace the conventional shading stage while faithfully respecting the underlying scene structure (Liang et al., 2025; Beisswenger et al., 2025; Zhang et al., 2026b). Because geometry, physics, and gameplay logic remain under the control of the game engine, this engine-plus-generative-renderer paradigm preserves the evolving world by construction, providing a practical foundation for real-time, prompt-controllable gameplay.

AlayaRenderer, the forward renderer of the Generative World Renderer system (Huang et al., 2026b), represents the state of the art in this direction. Trained on a large-scale corpus captured from visually complex AAA games, it renders scenes far beyond the visual complexity handled by previous G-buffer renderers while supporting text-guided appearance control. Players can dynamically restyle the game—changing lighting, atmosphere, or artistic style—without modifying geometry, materials, or gameplay logic. Despite its rendering quality, AlayaRenderer remains an offline renderer. Its 50-step denoising process dominates inference cost, while G-buffer encoding and VAE decoding introduce additional computational bottlenecks. Moreover, its bidirectional fixed-window formulation prevents autoregressive rollout over the unbounded G-buffer stream produced by a live game engine. As a result, prompt-controllable rendering cannot operate at interactive frame rates.

To bridge this gap, we introduce AlayaRenderer-Flash, a real-time deployment version of AlayaRenderer that preserves the original G-buffer and text-prompt interfaces while enabling interactive rendering through three complementary improvements. First, generation becomes autoregressive, allowing rendering history to be propagated across windows and enabling streaming over unbounded G-buffer sequences. Second, the original 50-step denoising schedule is distilled into a 4-step renderer, substantially reducing inference latency. Third, the computationally expensive G-buffer encoder and Wan VAE decoder (Wan et al., 2025) are replaced with distilled lightweight encoder and decoder networks. Together, these improvements preserve the rendering quality and controllability of AlayaRenderer while reducing inference cost to the point where prompt-controllable generative rendering can run alongside live gameplay instead of as an offline post-processing pipeline.

2 Method

Figure 2 illustrates the complete deployment pipeline, in which a game engine continuously exports synchronized G-buffer streams that are processed by AlayaRenderer-Flash to generate stylized RGB frames in real time. Internally, AlayaRenderer-Flash combines autoregressive streaming with hierarchical history compression, four-step diffusion, and lightweight distilled codecs to enable playback-rate rendering. The following subsections describe these designs in detail.

2.1 G-Buffer-Conditioned Latent Rendering

The base renderer, AlayaRenderer (Huang et al., 2026b), is built upon Wan 2.1 (Wan et al., 2025), a latent video diffusion model that denoises RGB video latents produced by the Wan causal 3D VAE while conditioning on text through cross-attention. The game engine provides five synchronized physical buffers—albedo, depth, metallic, normal, and roughness—describing scene geometry and material properties rather than final appearance. Each G-buffer channel is treated as a video stream and encoded by the same Wan causal 3D VAE as the RGB frames, yielding frame-aligned latent representations. The resulting features are

Figure 2 System overview of AlayaRenderer-Flash. During gameplay, the player independently controls game interaction through standard game inputs and visual appearance through text prompts. The game engine continuously updates the world state and exports synchronized G-buffers, which, together with the text prompt, are processed by AlayaRenderer-Flash to generate stylized RGB frames in real time. The rendered frames are immediately displayed to the player, forming a closed-loop interactive rendering system.

concatenated along the channel dimension to form the G-buffer condition g_k, which is concatenated with the noisy RGB latent before the first 3D patch embedding:

$$h_k = \text{PatchEmbed}([x_k, g_k]), \tag{1}$$

where x_k denotes the noisy RGB latent of chunk k, and g_k the corresponding G-buffer condition.

This conditioning strategy requires only a minimal modification to the pretrained backbone by widening the input projection of the first 3D patch embedding to accommodate the additional condition channels. The remainder of the diffusion transformer is left unchanged.

2.2 Autoregressive Streaming

AlayaRenderer performs bidirectional generation over a fixed-length window (21-latent-frame chunks) and therefore cannot support unbounded live rendering. AlayaRenderer-Flash instead operates autoregressively at the chunk level by partitioning the latent video into fixed-length chunks. In our experiments, each chunk contains four latent frames, and previously generated chunks are retained as history for subsequent generation. We organize the retained history into a three-tier compression hierarchy, with the compression level determined by its temporal distance from the current chunk. Recent history is preserved at full fidelity, while more distant history is progressively compressed into increasingly coarser representations, following the general idea of multi-scale history compression in recent work (Zhang et al., 2026a; Yuan et al., 2026). The first generated latent frame is additionally retained as a global appearance anchor by prepending it to the highest-fidelity history tier, allowing every chunk to attend to the stream's initial appearance. During denoising, all retained history is inserted into self-attention together with the current chunk as clean latent tokens, while the G-buffer latents remain the per-chunk conditioning signals for geometry and material properties.

To preserve prompt controllability over rollouts of arbitrary length, we further augment every self-attention layer with a dedicated text sink consisting of persistent key–value entries projected from the style-prompt embedding. Consequently, every chunk continuously attends to the prompt through self-attention in addition to the standard cross-attention conditioning.

The full encoding–decoding pipeline is likewise implemented causally: the VAE encoder carries its temporal feature cache across chunks so the G-buffer stream is encoded as one continuous sequence, and the decoder's temporal state persists across chunk calls, so that per-chunk streaming decoding is numerically consistent with a single-pass decode of the entire latent stream.

2.3 Few-Step Distillation

Reducing the number of denoising steps is one of the most effective ways to accelerate diffusion inference. We therefore target a four-step denoising schedule for deployment and use it throughout all experiments. However, directly compressing the original 50-step denoising process into four steps causes substantial quality degradation, since each denoising step must approximate a much longer trajectory than the teacher was trained to model.

To overcome this challenge, we adopt a three-stage few-step distillation pipeline consisting of guidance distillation, progressive step reduction, and Mean Flow Distillation (MFD) (Zhao et al., 2026). We find that the final distribution-refinement stage is highly sensitive to the student's initialization: directly applying MFD to an ODE-regression-initialized four-step student leads to unstable optimization. The first two stages therefore provide a stable initialization by progressively adapting the student to the target four-step budget before the final refinement stage.

Guidance Distillation. The autoregressive renderer introduced in Section 2.2 still relies on classifier-free guidance (CFG), requiring two network evaluations at every denoising step. We therefore distill the teacher's CFG-combined velocity field at the target guidance scale into the student (Meng et al., 2023). During this stage, the student retains the original 50-step denoising schedule while learning to reproduce the teacher's guided prediction with a single network evaluation. This converts explicit CFG into behavior encoded in the student weights, enabling all subsequent distillation stages to use the same single-pass formulation.

Progressive Step Reduction. To provide a stable initialization for the final distribution-refinement stage, we gradually reduce the denoising budget following Progressive Distillation (Salimans and Ho, 2022). Directly training the student under the target four-step budget leads to unstable optimization. Therefore, we sequentially train the guidance-distilled student using 32-, 16-, 8-, and finally 4-step schedules, allowing it to progressively adapt to increasingly larger denoising intervals and providing a stable initialization for the final refinement stage.

Mean Flow Distillation under self-rollout. After progressive distillation, the four-step student is further refined under the self-rollout training regime introduced in Self Forcing (Huang et al., 2026a). During this stage, the student conditions exclusively on its own previously generated chunks rather than ground-truth RGB history, matching the deployment setting and reducing the train–test discrepancy between teacher-forced training and autoregressive inference.

For the final distribution-refinement stage, we experimented with DMD-style adversarial distribution matching (Yin et al., 2024), but found it prone to unstable optimization and noticeable color artifacts in our setting. We therefore adopt MFD (Zhao et al., 2026), whose flow-based matching objective provides substantially more stable optimization for four-step refinement while maintaining comparable visual fidelity.

Finally, aggressive step reduction inevitably suppresses high-frequency details. To compensate for this loss, we attach lightweight GAN heads to intermediate transformer features and the final latent prediction (MFD+GAN). The adversarial loss is assigned a small weight to recover local textures without interfering with the MFD objective or destabilizing training.

2.4 Distilled Tiny Codecs

After reducing denoising to only a few diffusion steps, the remaining inference latency is dominated by the VAE modules at both ends of the pipeline. Specifically, encoding requires five independent Wan VAE passes, one for each G-buffer channel, while decoding relies on the streaming Wan VAE decoder. To further reduce latency, we distill both components into lightweight replacements: a shared tiny G-buffer encoder and a tiny temporal decoder.

Tiny decoder. The tiny decoder adopts the architecture of TAEHV (Bohan, 2025) and is initialized from its publicly released Wan 2.1 (Wan et al., 2025) pretrained weights. However, these pretrained weights do not reconstruct our game-domain content sufficiently well. We therefore further distill the tiny decoder from the frozen Wan VAE decoder. During training, both decoders receive the same latent representations produced

by the diffusion transformer, and the tiny decoder is optimized to reproduce the outputs of the Wan VAE decoder using a pixel reconstruction loss together with a small perceptual loss (Zhang et al., 2018).

Tiny G-buffer encoder. The input encoder is distilled in the same manner. Instead of performing five separate Wan VAE encoding passes, one for each G-buffer channel, we train a shared tiny G-buffer encoder that predicts the G-buffer condition g_k in Equation 1 with a single forward pass. The encoder is first distilled to match the latent representations produced by the frozen Wan VAE encoder, and is then jointly fine-tuned together with the entire renderer.

2.5 Real-Time Deployment

The techniques introduced above are integrated into a unified streaming renderer for real-time deployment. During inference, incoming G-buffer streams are continuously encoded by the distilled tiny encoder, rendered by the four-step diffusion model, and decoded into RGB frames by the stateful tiny decoder. All components operate autoregressively while maintaining persistent temporal states across chunk boundaries, enabling continuous rendering over unbounded input streams without restarting the rendering pipeline.

The complete inference pipeline is executed as a single optimized streaming system, minimizing runtime overhead while preserving temporal state throughout continuous execution. Together, the autoregressive streaming formulation, four-step diffusion model, and lightweight codecs enable playback-rate rendering suitable for interactive applications.

3 Experiments

Data & Experimental Setup. AlayaRenderer-Flash is trained and evaluated on an engine-captured *Black Myth: Wukong* dataset collected at a resolution of 1280×720 and 30 FPS. The dataset consists of 1,352 training clips and 131 test clips. Each frame contains an RGB reference together with five aligned G-buffer channels: albedo, depth, normal, roughness, and metallic. The data are collected using the same capture pipeline as the Generative World Renderer dataset (Huang et al., 2026b). Unless otherwise specified, all quantitative results are reported on the 131 test clips, each containing 150 frames (5 seconds), at a model input resolution of 832×448. All stages of training are performed on eight NVIDIA H200 GPUs.

Metrics. We evaluate rendering quality and deployment performance using the following complementary metrics.

- **Content preservation.** We report CLIP image-image similarity ($S_{\text{CLIP}-\text{I}}$) using the OpenCLIP ViT-L/14 image encoder (Radford et al., 2021). This metric measures semantic consistency between generated frames and their paired reference frames while remaining relatively insensitive to changes in lighting, color, and artistic style.
- **Temporal stability.** Temporal consistency is evaluated using warped temporal LPIPS ($\text{tLPIPS}_{\text{warp}}$) (Zhang et al., 2018). For each adjacent generated-frame pair, the previous frame is first aligned to the current frame using dense optical flow, and LPIPS is then computed between the aligned previous frame and the current frame. Lower values indicate less perceptual flicker.
- **Cross-window consistency.** To evaluate long-horizon streaming, we report Boundary MSE and Boundary SSIM across adjacent evaluation windows following the protocol of (Yang et al., 2026). Lower Boundary MSE and higher Boundary SSIM indicate smoother transitions between independently processed streaming windows.
- **Prompt controllability.** We quantify prompt controllability with a contrastive CLIP margin. For each generated frame sampled at a fixed stride, we compare its CLIP similarity to the prompt used for generation against its similarity to a contrast prompt from another test clip:

$$M_{\text{CLIP}} = \text{CLIP}(I, y^+) - \text{CLIP}(I, y^-),$$

  where y^+ is the conditioning prompt and y^- is the contrast prompt. The score is averaged over frames and clips. Higher values indicate stronger text control over rendered appearance, such as lighting,

Table 1 Comparison within the AlayaRenderer family. Bold marks the best value per column.

Method	Steps	$S_{\text{CLIP-I}}$ ↑	Boundary MSE ↓	Boundary SSIM ↑	$\text{tLPIPS}_{\text{warp}}$ ↓	M_{CLIP} ↑	FPS ↑	VRAM (GB) ↓
AlayaRenderer	50	0.836	0.0500	0.308	**0.124**	0.039	0.56	30.1
AlayaRenderer-AR	50	0.846	0.0433	0.358	0.197	0.030	1.53	22.6
AlayaRenderer-AR-distilled	4	0.843	0.0418	0.371	0.158	**0.045**	6.30	22.6
AlayaRenderer-Flash	4	**0.847**	**0.0406**	**0.430**	0.155	0.043	**31.54**	**16.2**

color, weather, and atmosphere. For prompt-switch evaluation, the previous prompt is used as y^-. Methods without text conditioning are marked as not applicable.

- **Deployment performance.** We report throughput (FPS) and peak GPU memory on an NVIDIA H200 GPU under identical runtime settings.

3.1 Progressive Design Analysis

Table 1 presents a controlled progression from the original AlayaRenderer to the final AlayaRenderer-Flash through four configurations: (1) the original AlayaRenderer, (2) autoregressive streaming with the original 50-step denoising schedule, (3) 4-step distilled rendering, and (4) lightweight deployment with the distilled encoder and decoder. All configurations are evaluated under the same 5-second protocol using identical G-buffer inputs, text prompts, and evaluation settings, allowing the contribution of each stage to be isolated.

Autoregressive reformulation. AlayaRenderer-AR isolates the effect of the proposed autoregressive streaming formulation while retaining the original 50-step denoising schedule. Compared with AlayaRenderer, it achieves higher content similarity ($S_{\text{CLIP}-\text{I}}$: 0.846 vs. 0.836) and improved boundary consistency (Boundary MSE: 0.0433 vs. 0.0500; Boundary SSIM: 0.358 vs. 0.308). Temporal stability decreases ($\text{tLPIPS}_{\text{warp}}$: 0.197 vs. 0.124), reflecting the more challenging autoregressive setting. Throughput increases from 0.56 FPS to 1.53 FPS, but remains far below the target playback rate, indicating that autoregressive streaming alone is insufficient for real-time deployment.

4-step distillation. Replacing the original 50-step denoising process with the proposed 4-step distilled model substantially improves computational efficiency while largely preserving rendering quality. Relative to the 50-step autoregressive renderer, throughput increases from 1.53 FPS to 6.30 FPS without increasing peak memory usage. Meanwhile, content similarity, boundary consistency, and prompt controllability remain comparable to, or slightly better than, the 50-step model. Temporal stability also improves considerably, with warped temporal LPIPS decreasing from 0.197 to 0.158. These results demonstrate that 4-step distillation substantially improves computational efficiency while maintaining high rendering quality.

Lightweight deployment. AlayaRenderer-Flash further replaces the original G-buffer encoder and Wan VAE decoder with their distilled lightweight counterparts while keeping the same four-step diffusion transformer. Consequently, this comparison isolates the contribution of the lightweight codec modules to deployment efficiency. The lightweight implementation increases throughput from 6.30 FPS to 31.54 FPS while reducing peak GPU memory from 22.6 GB to 16.2 GB. Meanwhile, rendering quality is maintained or slightly improved across content similarity, and boundary consistency. We attribute these gains to the lightweight encoder and decoder being distilled and fine-tuned on the target game domain. These results indicate that the lightweight codecs remove the remaining deployment bottleneck with negligible impact on rendering quality.

3.2 Capability and Performance Comparison

For a fair comparison, we retrain all external baselines on our training dataset using the same G-buffer benchmark whenever their implementations are publicly available and reproducible. Table 2 summarizes the capabilities of different methods, while Table 3 reports quantitative comparisons under the same 5-second evaluation protocol. RGB↔X (Zeng et al., 2024) performs independent per-frame rendering without temporal modeling. Consequently, it achieves the lowest content similarity and the highest FVD (Unterthiner et al., 2018) among the evaluated methods, indicating limited spatiotemporal coherence despite supporting

Table 2 Capability comparison with existing G-buffer rendering methods.

Method	G-buffer	Autoregressive	Prompt switch	Few-step	Length
RGB↔X (per-frame)	yes	no	yes	no	per-frame
FrameDiffuser (per-frame AR)	yes	yes	no	no	unbounded
AlayaRenderer-Flash	yes	yes	yes	yes	unbounded

Table 3 Quantitative comparison with external G-buffer rendering baselines under the fixed 5-second evaluation protocol. Bold indicates the best result.

Method	Window	$S_{\text{CLIP-I}} \uparrow$	FVD $\downarrow$	$\text{tLPIPS}_{\text{warp}} \downarrow$	$M_{\text{CLIP}} \uparrow$	FPS $\uparrow$	VRAM (GB)$\downarrow$
RGB↔X	5s	0.820	1031.3	0.305	0.027	1.30	**3.3**
FrameDiffuser	5s	0.844	650.6	0.440	n/a	0.31	3.5
AlayaRenderer-Flash	5s	**0.847**	**384.1**	**0.155**	**0.043**	**31.54**	16.2

text-guided appearance control. FrameDiffuser (Beisswenger et al., 2025) introduces autoregressive temporal modeling and improves content similarity over RGB↔X. However, it does not support prompt-controlled rendering and exhibits substantially weaker temporal stability ($\text{tLPIPS}_{\text{warp}} = 0.440$). Moreover, its inference speed is limited to 0.31 FPS, making real-time streaming deployment impractical.

We additionally retrain DiffusionRenderer (Liang et al., 2025) on our training dataset. Since DiffusionRenderer adopts a bidirectional formulation rather than autoregressive streaming, each 150-frame evaluation sequence is divided into multiple independent 25-frame windows, which are rendered offline and evaluated separately. This protocol allows DiffusionRenderer to exploit future context within each window, providing a favorable evaluation setting for bidirectional methods. Under this protocol, it achieves a higher $S_{\text{CLIP-I}}$ of 0.870 and a lower FVD of 335.5 than AlayaRenderer-Flash. However, its temporal stability remains comparable ($\text{tLPIPS}_{\text{warp}} = 0.156$), while inference runs at only 1.10 FPS, preventing real-time streaming deployment.

Overall, AlayaRenderer-Flash achieves the best trade-off between rendering quality and deployment efficiency. Although bidirectional methods can obtain stronger offline rendering quality under favorable evaluation protocols, AlayaRenderer-Flash is the only method that simultaneously supports causal streaming, prompt switching, few-step inference, and real-time interactive rendering while sustaining 31.54 FPS.

3.3 Qualitative Results

Figure 3 complements the quantitative evaluation with representative examples from two challenging sequences. The top block samples a snow scene across the full 5-second rollout to show long-horizon behavior; the bottom block samples a bamboo forest every five frames to expose short-term stability.

The qualitative observations are consistent with the quantitative results reported in Table 3. RGB↔X (Zeng et al., 2024) performs independent per-frame rendering without temporal modeling and therefore exhibits inconsistent illumination and visible temporal flicker throughout the sequence. FrameDiffuser (Beisswenger et al., 2025) preserves more scene structure than RGB↔X (Zeng et al., 2024) but accumulates appearance drift during long autoregressive rollouts, leading to noticeable changes in lighting, color tone, and geometry over time. In contrast, AlayaRenderer-Flash maintains stable illumination, scene geometry, and camera motion throughout the entire rollout while remaining closely aligned with the underlying G-buffer stream. These qualitative results demonstrate its ability to preserve long-horizon temporal consistency while maintaining high visual fidelity, consistent with the quantitative evaluation in Table 3.

3.4 Prompt Switching on Long Rollouts

Prompt-controllable rendering is a key capability of AlayaRenderer-Flash. Unlike offline renderers that require restarting the generation process after each prompt change, our method supports seamless prompt updates during continuous streaming inference. We further evaluate this capability on longer test sequences, each consisting of 637 frames rendered in a single streaming pass. During generation, the prompt is

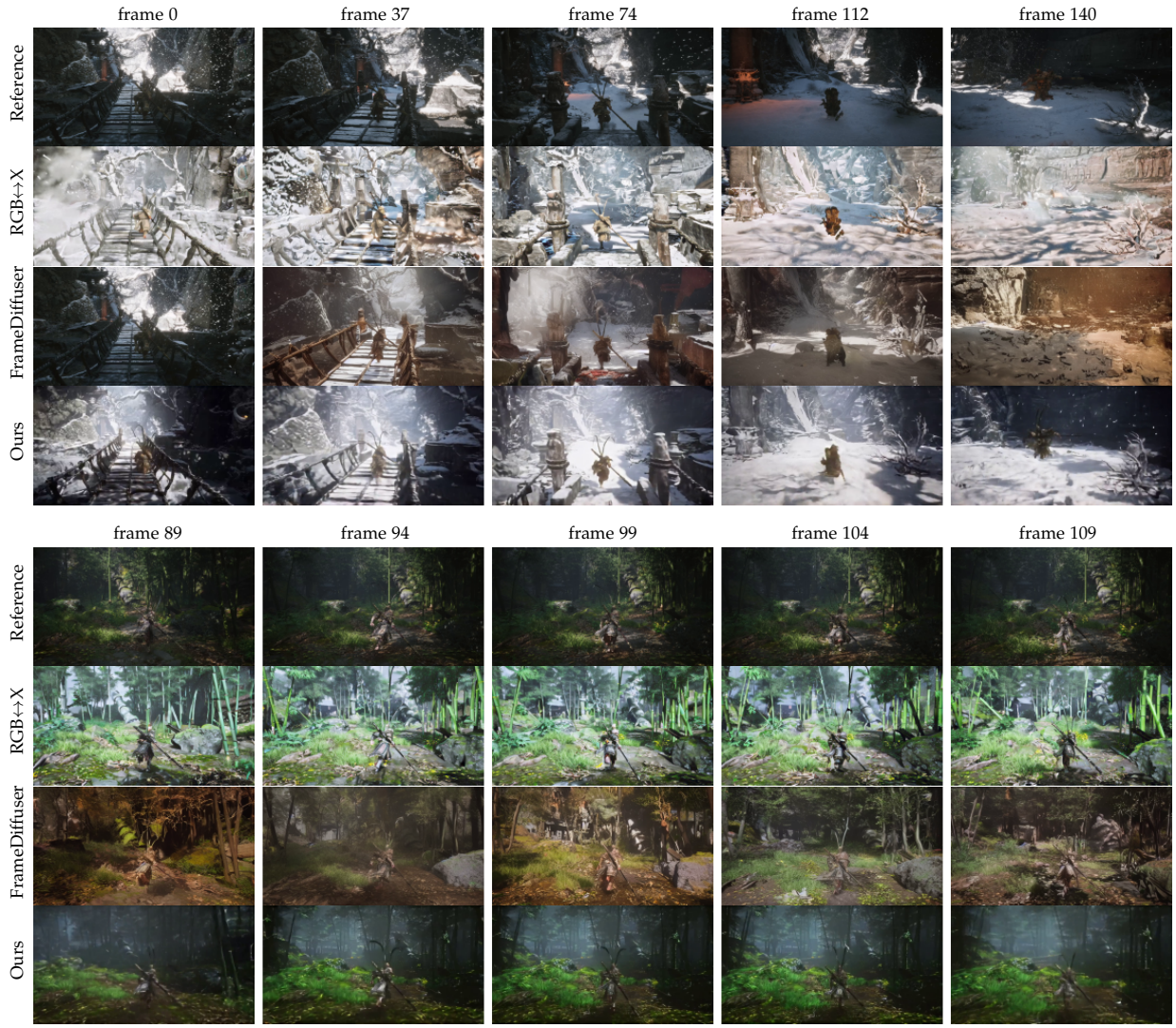

Figure 3 Qualitative comparison with external baselines on 5-second clips. Reference denotes the original RGB frames rendered by the game engine during data collection. Since prompt-conditioned rendering admits multiple valid outputs, these frames serve as visual references rather than ground-truth targets.

switched every five chunks, cycling through eight prompts spanning diverse artistic styles, lighting conditions, weather effects, and color palettes. Figure 4 shows a representative rollout. AlayaRenderer-Flash consistently follows each prompt transition while preserving scene geometry, character motion, and camera trajectory throughout the sequence. Transitions remain smooth without noticeable ghosting or boundary artifacts, demonstrating continuous prompt-controlled rendering over long streaming rollouts.

3.5 End-to-end Interactive Deployment

To validate practical deployment, we adapt AlayaRenderer-Flash to the open-source game SuperTuxKart (The SuperTuxKart Team, 2022). We first collect a target-domain dataset containing synchronized RGB frames and the same five G-buffer modalities used by the renderer, and fine-tune the model on this dataset. We then integrate the fine-tuned renderer into the live SuperTuxKart engine to form a complete real-time interactive rendering system. During gameplay, players control vehicle motion and camera viewpoints through standard game inputs while independently specifying the desired visual appearance using predefined style presets or arbitrary text prompts. The game engine continuously updates the world state and exports synchronized G-buffer streams, which are processed by AlayaRenderer-Flash to synthesize stylized RGB

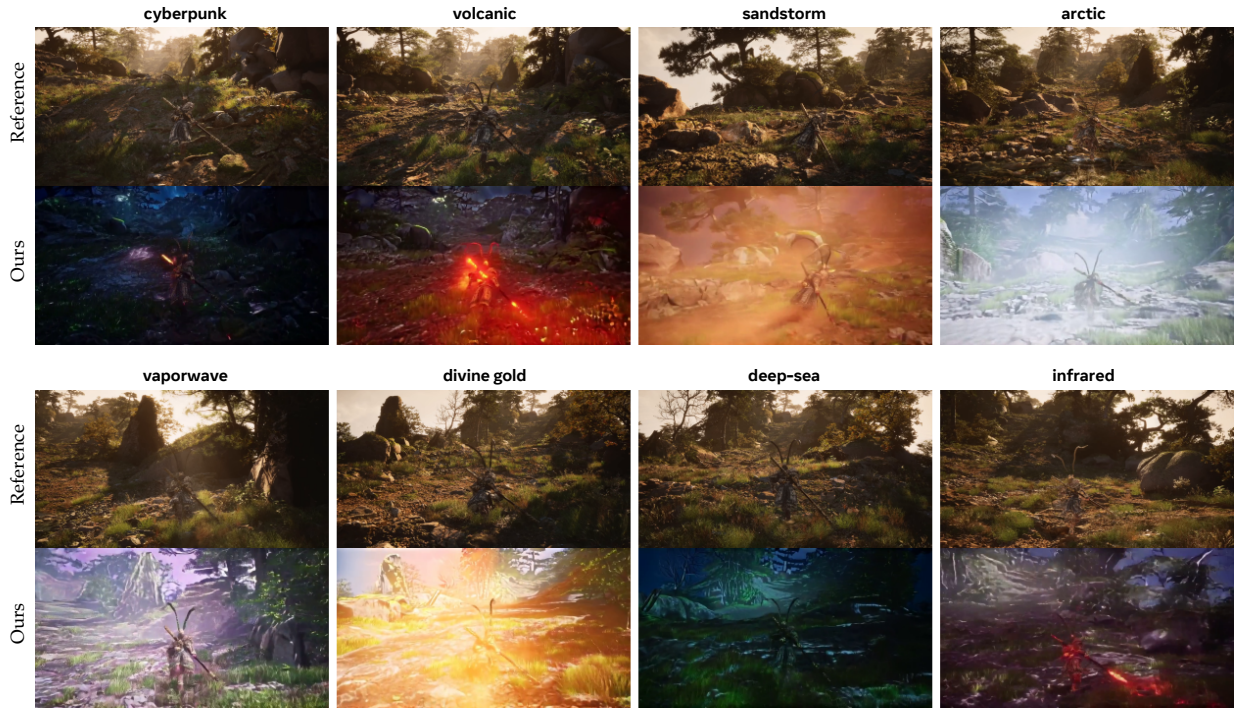

Figure 4 Prompt-controlled visual style transitions during a single long streaming rollout. Reference denotes the original RGB frames rendered by the game engine during data collection and is shown for visual comparison.

frames while preserving the engine-side gameplay logic, physics simulation, scene geometry, and camera motion. Prompts can be updated online without interrupting gameplay, enabling continuous visual-style changes during interaction.

The complete rendering pipeline is deployed on a single NVIDIA H200 GPU. Under this deployment setting, AlayaRenderer-Flash—including G-buffer encoding, four-step diffusion, and RGB decoding—runs at 31.54 FPS. Including engine-side G-buffer readback, data transfer, and display synchronization, the complete interactive system consistently sustains the target playback rate of 30 FPS during live gameplay. These results demonstrate that AlayaRenderer-Flash can be deployed as a practical interactive renderer, supporting continuous prompt-controlled world rendering during live gameplay.

4 Conclusion

We presented AlayaRenderer-Flash, a real-time deployment framework that transforms AlayaRenderer from an offline diffusion renderer into a streaming generative renderer for interactive applications. By combining autoregressive streaming, progressive four-step distillation, and lightweight codecs, AlayaRenderer-Flash enables real-time rendering while preserving the rendering quality, prompt controllability, temporal consistency, and structural fidelity of the original model.

Extensive experiments demonstrate that AlayaRenderer-Flash achieves a strong balance between rendering quality and deployment efficiency, supporting continuous prompt-controlled rendering over arbitrarily long streaming sequences at real-time speed. By integrating AlayaRenderer-Flash into a game engine, we further demonstrate a fully playable real-time game, validating the practical feasibility of streaming generative world rendering.

We hope this work serves as a practical step toward real-time generative world models and AI-native interactive environments.

References

Ole Beisswenger, Jan-Niklas Dihlmann, and Hendrik P. A. Lensch. FrameDiffuser: G-Buffer-conditioned diffusion for neural forward frame rendering. *arXiv preprint arXiv:2512.16670*, 2025.

Ollin Boer Bohan. TAEHV: Tiny autoencoder for hunyuan video. https://github.com/madebyollin/taehv, 2025.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. *Advances in Neural Information Processing Systems*, 38: 167283–167308, 2026a.

Zheng-Hui Huang, Zhixiang Wang, Jiaming Tan, Ruihan Yu, Yidan Zhang, Bo Zheng, Yu-Lun Liu, Yung-Yu Chuang, and Kaipeng Zhang. Generative World Renderer. *arXiv preprint arXiv:2604.02329*, 2026b.

Ruofan Liang, Zan Gojcic, Huan Ling, Jacob Munkberg, Jon Hasselgren, Chih-Hao Lin, Jun Gao, Alexander Keller, Nandita Vijaykumar, Sanja Fidler, and Zian Wang. Diffusion Renderer: Neural inverse and forward rendering with video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26069–26080, 2025.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14297–14306, 2023.

Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel. Deep shading: Convolutional neural networks for screen space shading. In *Computer Graphics Forum*, volume 36, pages 65–78. Wiley Online Library, 2017.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 197–206, 1990.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

The SuperTuxKart Team. SuperTuxKart: A 3d open-source kart racing game. https://supertuxkart.net, 2022.

Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics*, 38(4):1–12, 2019.

Thomas Unterthiner, Sjoerd Van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Jing Yang, Mayoore Jaiswal, Zian Wang, Steven Zeng, Rochelle Pereira, Yajie Zhao, and Jianyuan Min. HorizonRelight: Relighting long-horizon videos consistently via diffusion transformers. *arXiv preprint arXiv:2606.29095*, 2026.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024.

Shenghai Yuan, Yuanyang Yin, Zongjian Li, Xinwei Huang, Xiao Yang, and Li Yuan. Helios: Real real-time long video generation model. *arXiv preprint arXiv:2603.04379*, 2026.

Zheng Zeng, Valentin Deschaintre, Iliyan Georgiev, Yannick Hold-Geoffroy, Yiwei Hu, Fujun Luan, Ling-Qi Yan, and Miloš Hašan. RGB↔X: Image decomposition and synthesis using material- and lighting-aware diffusion models. In *ACM SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024.

Lvmin Zhang, Shengqu Cai, Muyang Li, Gordon Wetzstein, and Maneesh Agrawala. Frame context packing and drift prevention in next-frame-prediction video diffusion models. *Advances in Neural Information Processing Systems*, 38:30546–30566, 2026a.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

Shenghao Zhang, Runtao Liu, Christopher Schroers, and Yang Zhang. RenderFlow: Single-step neural rendering via flow matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 40602–40611, 2026b.

An Zhao, Shengyuan Zhang, Zhongjian Sun, Yixiang Zhou, Zejian Li, Ling Yang, Tianrun Chen, and Lingyun Sun. Mean flow distillation: Robust and stable distillation for flow matching models. *arXiv preprint arXiv:2606.11155*, 2026.