

UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Toward Generative Rendering: Controllable Image Synthesis with High Photorealism and Low Cost

Permalink

<https://escholarship.org/uc/item/7zv8h7ff>

ISBN

9798186382218

Author

Zeng, Zheng

Publication Date

2026-06-14

Peer reviewed|Thesis/dissertation

University of California
Santa Barbara

Toward Generative Rendering: Controllable Image Synthesis with High Photorealism and Low Cost

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Zheng Zeng

Committee in charge:

Professor Lingqi Yan, Chair, Mohamed bin Zayed University of Artificial In-
telligence
Professor Tobias Höllerer
Professor Shiyu Chang

June 2026

The Dissertation of Zheng Zeng is approved.

Professor Tobias Höllerer

Professor Shiyu Chang

Professor Lingqi Yan, Committee Chair

April 2026

Acknowledgements

I am deeply grateful to my advisor, Professor Lingqi Yan, for his guidance, support, and trust throughout my Ph.D. His insight shaped the way I think about research, and his encouragement gave me the freedom to explore the problems that became this dissertation.

I would also like to thank my committee members, Professor Tobias Höllerer and Professor Shiyu Chang, for their time, feedback, and support.

I am very grateful to my industry mentors Edward Liu, Miloš Hašan, and Daqi Lin for their guidance, collaboration, and support during my internships and research projects. I also thank my many other collaborators, whose ideas and effort made this work possible.

I would like to thank my friends Songyin Wu, Chuyan Zhang, Yuxuan Yin, Tao Huang, Zilin Xu, Shanxiu He, Yangchao Wu, Lifan Wu, and my other labmates for their friendship, help, and many memorable conversations throughout these years.

Curriculum Vitæ

Zheng Zeng

Education

- 2021–2026 Ph.D. in Computer Science, University of California, Santa Barbara.
Advisor: Professor Lingqi Yan.
- 2014–2021 B.S. and M.S. in Software Engineering, Shandong University.

Research Experience

- 2025–present Research Intern, NVIDIA Real-time Rendering Research, Santa Clara, CA.
- 2024–2025 Research Intern, NVIDIA Real-time Rendering Research, Redmond, WA.
- 2023–2024 Research Scientist/Engineer Intern, Adobe Research, San Jose, CA.
- 2022–2023 Research Intern, NVIDIA Applied Deep Learning Research, Santa Clara, CA.

Publications

- 2026 Zheng Zeng, Marco Salvi, Lifan Wu, Jan Novák, Daqi Lin, Saeed Hadadan, Yichen Sheng, Robert Pottorff, Shiqiu Liu, Ravi Ramamoorthi, Ling-Qi Yan, and Miloš Hašan. *RGBX-Next: Towards Realistic Generative Rendering from G-Buffers*.
- 2026 Bowen Xue, Saeed Hadadan, Zheng Zeng, Fabrice Rousselle, Zahra Montazeri, and Miloš Hašan. *VideoNeuMat: Neural Material Extraction from Generative Video Models*. ACM SIGGRAPH 2026 Conference Track.
- 2026 Jon Hasselgren, Zheng Zeng, Miloš Hašan, and Jacob Munkberg. *VideoMatGen: PBR Materials through Joint Generative Modeling*. CVPR 2026 Findings.
- 2025 Zheng Zeng, Markus Kettunen, Chris Wyman, Lifan Wu, Ravi Ramamoorthi, Ling-Qi Yan, and Daqi Lin. *ReSTIR PG: Path Guiding with Spatiotemporally Resampled Paths*. ACM SIGGRAPH Asia 2025 Conference Track.
- 2025 Youxin Xing, Zheng Zeng, Youyang Du, Lu Wang, and Beibei Wang. *Diffusion-Guided Relighting for Single-Image SVBRDF Estimation*. ACM SIGGRAPH Asia 2025 Conference Track.
- 2025 Xiaolong Wu, Zheng Zeng, Junqiu Zhu, and Lu Wang. *Reshadable Level-of-Detail Impostors for Fast Distant Objects Rendering*. Eurographics Symposium on Rendering 2025, CGF Track.

- 2025 Yaoyi Bai, Songyin Wu, Zheng Zeng, Beibei Wang, and Ling-Qi Yan. *BSDF Importance Baking: A Lightweight Neural Solution to Importance Sampling General Parametric BSDFs*. Computer Graphics Forum, 2025.
- 2024 Zheng Zeng, Valentin Deschaintre, Iliyan Georgiev, Yannick Hold-Geoffroy, Yiwei Hu, Fujun Luan, Ling-Qi Yan, and Miloš Hašan. *RGB $\leftrightarrow$ X: Image Decomposition and Synthesis Using Material- and Lighting-Aware Diffusion Models*. ACM SIGGRAPH 2024 Conference Track.
- 2023 Songyin Wu, Sungye Kim, Zheng Zeng, Deepak Vembar, Sangeeta Jha, Anton Kaplanyan, and Ling-Qi Yan. *ExtraSS: A Framework for Joint Spatial Super Sampling and Frame Extrapolation*. ACM SIGGRAPH Asia 2023 Conference Track.
- 2023 Zheng Zeng, Zilin Xu, Lu Wang, Lifan Wu, and Ling-Qi Yan. *Ray-aligned Occupancy Map Array for Fast Approximate Ray Tracing*. Eurographics Symposium on Rendering 2023, CGF Track.
- 2022 Zilin Xu, Zheng Zeng, Lifan Wu, Lu Wang, and Ling-Qi Yan. *Lightweight Neural Basis Functions for All-Frequency Shading*. ACM SIGGRAPH Asia 2022.
- 2021 Zheng Zeng, Shiqiu Liu, Jinglei Yang, Lu Wang, and Ling-Qi Yan. *Temporally Reliable Motion Vectors for Real-time Ray Tracing*. Eurographics 2021, CGF Track; Ray Tracing Gems II.
- 2020 Zheng Zeng, Lu Wang, Beibei Wang, Chun-Meng Kang, and Yan-ning Xu. *Denoising Stochastic Progressive Photon Mapping Renderings Using a Multi-Residual Network*. Journal of Computer Science and Technology, CVM 2020.
- 2020 Yezi Zhao, Beibei Wang, Yanning Xu, Zheng Zeng, Lu Wang, and Nicolas Holzschuch. *Joint SVBRDF Recovery and Synthesis From a Single Image Using an Unsupervised Generative Adversarial Network*. Eurographics Symposium on Rendering 2020, DL-only Track.

Abstract

Toward Generative Rendering: Controllable Image Synthesis with High Photorealism
and Low Cost

by

Zheng Zeng

Photorealistic image synthesis has long been a central goal of computer graphics. Classical rendering pursues this goal by simulating light transport on an explicit scene description: it delivers precise artistic control and consistency, but falls short of ultimate photorealism and depends on costly assets and expensive light transport. Generative image and video models pursue the same goal differently. They learn the distribution of real photographs and videos and synthesize new samples directly, achieving high photorealism, yet they lack precise control and consistency.

In this dissertation, we combine the strengths of both approaches through *generative rendering*, using *G-buffers* – per-pixel scene quantities such as albedo, normals, roughness, metallicity, depth, and lighting – as a structured control interface between a rendering pipeline and a learned generative model. G-buffers are structured enough to carry precise artistic control, yet compact enough to leave a generative model room to synthesize the photorealism that is expensive to author and simulate. We call the resulting framework $\text{RGB} \leftrightarrow \text{X}$, where RGB denotes the final pixel image and X denotes the stack of G-buffer channels. The framework consists of two paired models: a forward model $\text{X} \rightarrow \text{RGB}$ that performs generative rendering by synthesizing an RGB image from G-buffers, and an inverse model $\text{RGB} \rightarrow \text{X}$ that performs inverse rendering by estimating G-buffers from an RGB image.

In the first part, we develop an image-domain instance of $\text{RGB} \leftrightarrow \text{X}$ that validates G-

buffers as a practical intermediate medium through which users can controllably generate the images they want. The $X \rightarrow \text{RGB}$ model renders photorealistic images from G-buffer inputs, giving control over base color (albedo), geometry (normal), materials (roughness and metallicity), and lighting, for which we use diffuse irradiance as the control representation. Input G-buffers can be partial: users fix only the properties they care about and let $X \rightarrow \text{RGB}$ creatively fill in the rest. The paired $\text{RGB} \rightarrow X$ model estimates the same G-buffers from real RGB images, closing the loop between forward and inverse generative rendering and enabling editing applications such as material replacement and object insertion. Because no dataset pairs real photographs with G-buffers, both $X \rightarrow \text{RGB}$ and $\text{RGB} \rightarrow X$ in this first part are trained primarily on synthetic renderings. $X \rightarrow \text{RGB}$ therefore inherits a synthetic look that bounds its photorealism.

In the second part, we push generative rendering toward high photorealism by training $X \rightarrow \text{RGB}$ directly on real data. The obstacle is the paired data bottleneck identified above. We resolve this by annotating large collections of real videos with our own $\text{RGB} \rightarrow X$ model to produce paired G-buffers and RGB images. We also label each training clip as real or synthetic, so that at inference time classifier-free guidance using these labels can push $X \rightarrow \text{RGB}$ generation toward photorealism. We introduce a set of new finetuning and streaming techniques on a pretrained diffusion transformer video backbone that accelerate convergence while avoiding catastrophic forgetting, yielding a unified $\text{RGB} \leftrightarrow X$ framework for both forward and inverse rendering of images, videos, and streams, with flexible lighting control and long-sequence streaming support.

Together, these results point toward image synthesis systems that retain the structure and controllability of classical rendering while achieving the photorealism learned by generative models from real data.

Contents

Curriculum Vitae	iv
Abstract	vi
1 Introduction	1
2 Background	6
2.1 Photorealistic Rendering	6
2.2 Inverse Rendering	11
2.3 Diffusion Generative Models	13
2.4 Summary	18
3 Controllable Image Synthesis from G-buffers	19
3.1 G-buffers and datasets	22
3.2 The RGB→X model	25
3.3 The X→RGB model	27
3.4 Results	30
3.5 Summary	34
4 Learning Photorealistic Generative Rendering from Real Data	39
4.1 Repurposing DiT video models for our tasks	42
4.2 RGB→X models for inverse rendering	46
4.3 X→RGB models for realistic generative rendering	50
4.4 Streaming models	56
4.5 Unified model	59
4.6 Results	61
4.7 Summary	72
5 Conclusion and Future Work	76
Bibliography	80

Chapter 1

Introduction

Image synthesis is a central problem of computer graphics. Its aim is to produce images that convey a desired scene or story. The dominant answer to this goal is *rendering*: given a description of geometry, materials, lights, and a camera, a rendering algorithm simulates how light interacts with the scene, and outputs an image. Rendering is today the engine behind video games, visual effects in film and animation, product visualization, architectural design, simulators for autonomous systems, and virtual and augmented reality. Among the many goals of rendering, one of the most important is *photorealistic rendering*: producing images that are indistinguishable from real photographs.

Rendering has clear strengths. First, it operates on an explicit scene description. Geometry, materials, lighting, and the camera are each represented as a separate, editable component. Light transport is simulated on top of these components, as reviewed in Chapter 2. Because every component is an exposed knob, artists and production pipelines enjoy precise, reproducible control. A sofa moved one meter to the left moves exactly one meter. A wall painted red stays the same red across every frame. This same structure also delivers consistency across time and across viewpoints. The underlying 3D representation is shared across frames and camera paths, so objects keep their identity, geometry, and materials as the view changes. And real-time rendering has been engineered so aggressively that modern game engines can render visually rich, dynamic

scenes at 60 frames per second on a single consumer GPU.

Rendering also has clear limits. Even state-of-the-art renderings can be distinguished from real photographs under careful inspection. The reason is that the theory behind rendering is divided into separate modules such as cameras, intersections, light distribution, visibility, light scattering at surfaces, light transport, and ray propagation. Each module is developed and optimized in isolation against its own proxy target. The pipeline as a whole is never optimized end-to-end against real photographs. Small errors at each stage accumulate into a visible gap between rendered images and real ones. The consequences are felt across applications. Games and films still exhibit a perceptibly synthetic appearance. Robotics and autonomous-driving simulators suffer a persistent sim-to-real gap that limits how much of what is learned in simulation transfers to the real world. Moreover, even the photorealism that rendering does achieve is costly to produce. Real surfaces carry irregularities, wear, dust, sub-pixel geometry, and subtle imperfections that are time-consuming and expensive for artists to author. Real lighting is even more demanding. Light bounces many times between surfaces, scatters through participating media, and forms complex focusing patterns such as caustics. Accurately capturing all of these effects requires tracing many light paths per pixel, and remains prohibitively expensive at production quality. All of this translates into high human labor for content authoring, high storage and bandwidth demands, and high computational cost on consumer devices.

A different approach to image synthesis has emerged: *generative image and video models*. Rather than simulating a scene, these models learn the distribution of real photographs and videos directly. They then synthesize new samples. As reviewed in Chapter 2, this family of models does not attempt to obey physics explicitly. Instead, it absorbs a huge amount of real visual data and reproduces the statistics of that data at test time. Recent generative image and video models produce appearance detail, material

variation, and lighting nuance that are difficult or impossible to author by hand.

Generative models also have clear limits. First, they lack explicit control for image synthesis tasks. Common control interfaces such as text prompts and reference images cannot reliably specify a particular material, the position of a particular light, the exact geometry of a particular object, or the camera path of a particular shot. Second, they lack consistency across frames. Identity, materials, and lighting drift from frame to frame in ways that are unacceptable for production use. Third, generative image and video models are expensive to develop and costly to run. Developing a state-of-the-art model requires hundreds of millions to billions of captioned examples and tens of thousands of GPU-days. Inference still requires GPU clusters well beyond consumer-level hardware. Acceleration techniques such as distillation and quantization have narrowed the gap, but not closed it. A traditional rendering pipeline, purpose-built for image synthesis, delivers images at millisecond-scale per-frame latency; generative models remain far from this regime.

Classical rendering and generative models therefore offer complementary strengths. Rendering gives control and consistency, but bounded photorealism at a high authoring and compute cost. Generative models give learned realism from real data, but lack control and consistency. In this dissertation, we combine the two into a single framework, which we call *generative rendering*. The key idea is to use *G-buffers* as a structured control interface between a traditional rendering pipeline and a learned generative model. G-buffers are the intermediate, image-aligned buffers that rendering pipelines already produce: albedo, normals, roughness, metallicity, depth, irradiance, and direct lighting. They are structured and physically interpretable enough to carry precise control. At the same time, they are compact and under-specified enough to leave a generative model room to synthesize the texture, imperfection, and high-frequency realism that rendering struggles to produce directly.

Building on this idea, we develop a family of models we call $\text{RGB} \leftrightarrow \text{X}$. Here RGB denotes the final pixel image, and X denotes the stack of G-buffers that describe the scene: geometry, materials, and lighting. The $\text{X} \rightarrow \text{RGB}$ direction performs generative rendering, and the $\text{RGB} \rightarrow \text{X}$ direction performs inverse rendering. Together, the two directions target the central goal of this dissertation: controllable image synthesis with high photorealism and low cost. $\text{RGB} \leftrightarrow \text{X}$ makes two main contributions toward this goal. First, in Chapter 3 we validate G-buffers as a practical intermediate medium through which users can controllably generate the images they want. We introduce a framework in which a generative model renders realistic images from G-buffer inputs. Users can control base color (albedo), surface geometry (normal), surface materials (roughness and metallicity), and lighting, for which we use diffuse irradiance as the control representation. Input G-buffers can also be partial: users fix only the properties they care about, and let the generative model creatively fill in the rest. A paired inverse model estimates the same G-buffers from real RGB images. Together, the forward and inverse models close the loop between generative rendering and inverse rendering, and enable editing applications such as material replacement and object insertion, without reconstructing a full 3D scene. Because no public dataset pairs real photographs with ground-truth G-buffers, both models in Chapter 3 are trained primarily on synthetic renderings assembled from several heterogeneous sources; the forward model consequently inherits a residual synthetic look that bounds the photorealism it can reach, and that bound is what the second contribution sets out to remove.

Second, in Chapter 4 we push generative rendering toward high photorealism by training the forward model directly on real data. The obstacle is the paired data bottleneck just noted – real videos carry no G-buffer annotations. We close this loop by turning our own inverse model into an annotator: we run it on large collections of real videos to produce paired G-buffers and RGB images that no existing dataset supplies.

We also label each training clip as real or synthetic. At inference time, classifier-free guidance using these labels pushes the forward model toward photorealism. To adapt a pretrained diffusion transformer video backbone to this setting, we introduce a set of new techniques: frame-wise concatenation with QK type embedding and clean input tokens, X-patchify for packing multiple G-buffer inputs into a single token set, and a streaming recipe that combines teacher forcing, self forcing, and long-context tokens. Together, these techniques accelerate finetuning convergence, avoid catastrophic forgetting of the base model’s priors, and mitigate drift over long sequences. The result is a unified framework that supports both forward and inverse rendering of images, videos, and streams, with flexible lighting control.

In what follows, Chapter 2 reviews the background needed for the rest of the dissertation: photorealistic rendering and its G-buffers, inverse rendering of these G-buffers, and generative image and video models. Chapter 3 then develops the first contribution, establishing G-buffers as a control interface for generative rendering. Chapter 4 develops the second contribution, pushing generative rendering toward high photorealism by learning directly from real data. Chapter 5 summarizes the work and discusses open directions.

Chapter 2

Background

This chapter collects the rendering and generative-model machinery used by the rest of the dissertation. The central object is the G-buffer stack $\mathbf{X}$: a set of image-aligned, physically interpretable channels that can either be rendered from a scene or estimated from pixels. Chapter 3 uses $\mathbf{X}$ in the image domain with latent diffusion models. Chapter 4 then lifts the same interface to video diffusion transformers, where RGB frames and G-buffer frames become typed latent tokens inside one model. The goal of this chapter is therefore twofold: first, to define the physical quantities that make G-buffers meaningful; second, to fix the diffusion, latent-video, and token notation that makes the mathematical narrative of Chapter 4 concise.

2.1 Photorealistic Rendering

Physically based rendering synthesizes an image by evaluating how light interacts with an explicitly authored scene. A complete treatment is given by [1]; here we recall only the radiometric quantities and equations needed later.

Table 2.1: Radiometric quantities used throughout the dissertation.

Quantity	Symbol	Definition	Unit
Radiant flux	Φ	energy per unit time	W
Radiance	L	flux per area per solid angle	$\text{W m}^{-2} \text{sr}^{-1}$
Irradiance	E	hemispherical flux per area	W m^{-2}
BRDF	f_r	outgoing radiance per incident irradiance	sr^{-1}

2.1.1 Scene representation

A classical scene description contains geometry, materials, lights, and a camera. Geometry specifies where surfaces are. Materials specify how each surface reflects or emits light. Lights specify how radiant energy enters the scene, either through local emitters or distant environment maps. The camera maps outgoing radiance to pixels. This explicit representation gives rendering its controllability and temporal consistency: changing a material parameter or light source has a predictable effect. It also explains the authoring and computation cost that motivates the learned renderers developed in this dissertation.

2.1.2 The rendering equation

At an opaque surface point $\mathbf{x}$ with normal $\mathbf{n}$, the outgoing radiance toward direction ω_o is

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i. \quad (2.1)$$

The emitted term L_e accounts for light produced at $\mathbf{x}$. The integral accounts for incident radiance L_i arriving from the upper hemisphere Ω , modulated by the BRDF f_r and the foreshortening cosine. Because L_i is itself the outgoing radiance of other scene points, Equation (2.1) is recursive. Monte Carlo path tracing estimates this recursive light transport by sampling random light paths. It is unbiased in the limit, but noise-free photorealism can require many samples per pixel and careful scene authoring.

2.1.3 Material models

The BRDF f_r is commonly represented by a microfacet model [2]. A rough surface is modeled as a distribution of small specular facets with half-vector ω_h and normal distribution function $D(\omega_h)$. The GGX distribution is

$$D_{\text{GGX}}(\omega_h) = \frac{\alpha^2}{\pi ((\omega_h \cdot \mathbf{n})^2 (\alpha^2 - 1) + 1)^2}, \quad (2.2)$$

where α controls microscopic roughness. The corresponding microfacet BRDF is

$$f_r(\mathbf{x}, \omega_i, \omega_o) = \frac{F(\omega_i, \omega_h) D(\omega_h) G(\omega_i, \omega_o, \omega_h)}{4(\omega_i \cdot \mathbf{n})(\omega_o \cdot \mathbf{n})}. \quad (2.3)$$

Here F is the Fresnel term and G accounts for geometric shadowing and masking.

Modern production and real-time renderers expose this model through the metallic-roughness parametrization [3]: base color $\mathbf{a} \in \mathbb{R}^3$, roughness $\mathbf{r} \in [0, 1]$, and metallicity $\mathbf{m} \in [0, 1]$. The roughness convention is $\alpha = \mathbf{r}^2$. The metallicity linearly blends a dielectric lobe, whose diffuse color is $\mathbf{a}$, with a metallic lobe, whose specular color is tinted by $\mathbf{a}$. This is the material family used throughout Chapters 3 and 4.

2.1.4 Lighting representations

The full incident radiance field $L_i(\mathbf{x}, \omega_i)$ is too large to use directly as a dense control signal. We therefore use lower-dimensional lighting summaries. The first is diffuse irradiance,

$$\mathbf{E}(\mathbf{x}) = \int_{\Omega} L_i(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i. \quad (2.4)$$

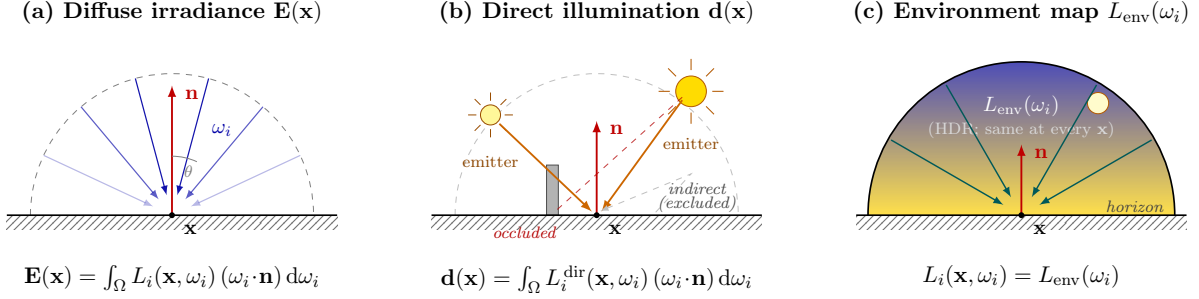


Figure 2.1: Lighting summaries used in this dissertation. Diffuse irradiance $\mathbf{E}$ integrates incoming radiance over the hemisphere. Direct illumination $\mathbf{d}$ keeps only first-bounce paths from light sources. An environment map stores distant incident radiance as a function of direction.

For a Lambertian surface, this reduces the rendering equation to

$$L_o(\mathbf{x}) = \frac{\mathbf{a}(\mathbf{x})}{\pi} \mathbf{E}(\mathbf{x}). \quad (2.5)$$

Thus $\mathbf{E}$ captures the view-independent lighting needed by diffuse shading. It is low-frequency and loses directional information, but it is compact and physically meaningful.

The second summary, used in Chapter 4, is albedo-free direct illumination,

$$\mathbf{d}(\mathbf{x}) = \int_{\Omega} L_i^{\text{dir}}(\mathbf{x}, \omega_i) (\omega_i \cdot \mathbf{n}) d\omega_i, \quad (2.6)$$

where L_i^{dir} includes only first-bounce light from emissive sources and environment lighting. It is cheaper to compute than full multi-bounce irradiance, captures sharp direct-light structure, and remains orthogonal to albedo because color multiplication has not yet been applied.

The third representation is an environment map $L_{\text{env}}(\omega_i)$, an HDR image indexed by direction. It represents distant incoming radiance and is standard in both classical rendering and relighting systems.

Table 2.2: G-buffer channels $\mathbf{X}$ used in the dissertation. Each channel is aligned with the RGB image $\mathbf{I}$. In Chapter 4, roughness and metallicity are often grouped into a single material channel $\mathbf{m}$.

Symbol	Quantity	Shape
$\mathbf{I}$	RGB image	$H \times W \times 3$
$\mathbf{a}$	Albedo or base color	$H \times W \times 3$
$\mathbf{n}$	Surface normal	$H \times W \times 3$
$\mathbf{r}$	Surface roughness	$H \times W$
$\mathbf{m}$	Metallicity	$H \times W$
$\mathbf{E}$	Diffuse irradiance	$H \times W \times 3$
$\mathbf{d}$	Direct illumination	$H \times W \times 3$
$\mathbf{z}$	Depth	$H \times W$
$\mathbf{m}$	Material bundle	$H \times W \times C_m$

2.1.5 G-buffers

G-buffers are image-aligned buffers produced by rendering pipelines before final shading. They store local scene quantities such as albedo, normals, depth, material parameters, and lighting summaries. They are the interface around which this dissertation is built: structured enough to support precise control, but incomplete enough to leave a learned model room to synthesize missing high-frequency appearance.

The table entries connect directly to the rendering equations above. The triplet $(\mathbf{a}, \mathbf{r}, \mathbf{m})$ parametrizes f_r in Equation (2.3); $\mathbf{n}$ is the surface normal in Equation (2.1); $\mathbf{z}$ recovers a surface point $\mathbf{x}$ given camera intrinsics; and $\mathbf{E}$ and $\mathbf{d}$ summarize L_i . The learned renderers in later chapters replace explicit evaluation of Equation (2.1) with a conditional generative model, but they keep these physically interpretable quantities as the control language.

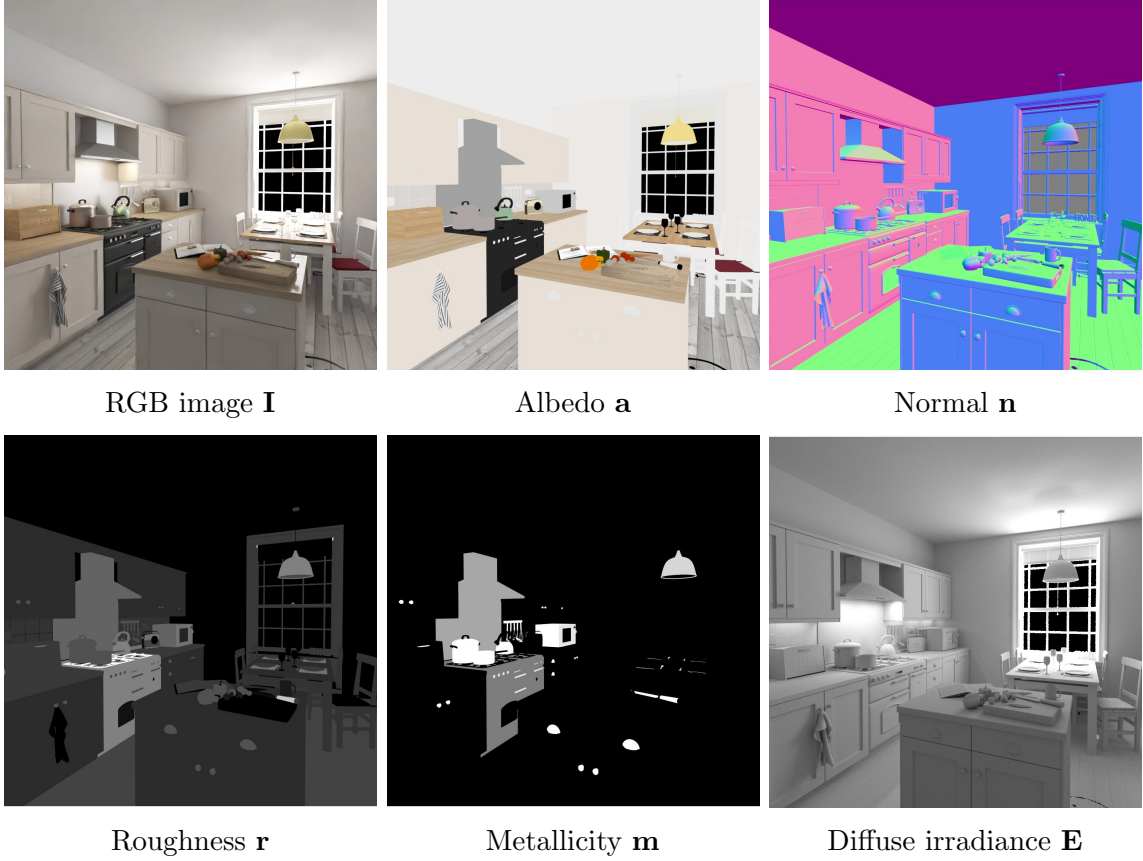


Figure 2.2: A rendered kitchen scene and its core G-buffer channels. These aligned buffers form the structured control representation used by $\text{RGB} \leftrightarrow \mathbf{X}$.

2.2 Inverse Rendering

Inverse rendering asks for scene quantities that explain an observed image. A full inverse-rendering solution would recover geometry, materials, lighting, and camera parameters under Equation (2.1). This is highly under-constrained: many scenes can produce the same pixels. In this dissertation we use a per-pixel form of inverse rendering, where the recovered quantities are the G-buffer channels $\mathbf{X}$ rather than a complete 3D scene:

$$\text{RGB} \rightarrow \mathbf{X} : \quad \mathbf{I} \mapsto \mathbf{X}. \quad (2.7)$$

This formulation is also called intrinsic-channel estimation in the inverse-rendering literature. It is still ambiguous, especially for albedo-lighting separation and material estimation, but it produces the exact representation needed by the forward direction,

$$\mathbf{X} \rightarrow \text{RGB} : \quad \mathbf{X} \mapsto \mathbf{I}. \quad (2.8)$$

The two directions are paired throughout the dissertation. Chapter 3 trains them primarily on synthetic renderings. Chapter 4 uses an improved $\text{RGB} \rightarrow \mathbf{X}$ model to annotate real videos with estimated G-buffers, then trains $\mathbf{X} \rightarrow \text{RGB}$ on those real RGB frames.

2.2.1 Intrinsic image decomposition

Classical intrinsic image decomposition splits an image into reflectance and shading, usually in the simplified form $\mathbf{I} \approx \mathbf{a} \odot \mathbf{s}$, where $\mathbf{s}$ is a shading image. Our G-buffer formulation generalizes this split: $\mathbf{a}$ remains the reflectance-like channel, while $\mathbf{n}$, $\mathbf{r}$, $\mathbf{m}$, $\mathbf{z}$, $\mathbf{E}$, and $\mathbf{d}$ expose additional geometric, material, and lighting factors. The ambiguity of the classical problem remains, so learned priors are essential.

2.2.2 Generative priors for inverse rendering

Modern generative models provide a prior over natural images, video motion, and plausible scene structure. In $\text{RGB} \rightarrow \mathbf{X}$, this prior regularizes ambiguous decompositions; in $\mathbf{X} \rightarrow \text{RGB}$, it fills appearance details absent from the G-buffer stack. The rest of this chapter formalizes the diffusion and video-DiT priors used to do this.

2.3 Diffusion Generative Models

Diffusion models generate data by learning to reverse a corruption process. Let $\mathbf{x}_0$ be a clean latent or pixel sample and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ be Gaussian noise. In DDPM notation [4], a fixed forward process is

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{\alpha_t}\mathbf{x}_{t-1}, (1 - \alpha_t)\mathbf{I}), \quad (2.9)$$

with the closed-form sample

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}. \quad (2.10)$$

The model learns a reverse denoising direction. In the original noise-prediction form, the loss is

$$\mathcal{L}_{\boldsymbol{\epsilon}} = \mathbb{E}_{\mathbf{x}_0, t, \boldsymbol{\epsilon}} [\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_{\theta}(\mathbf{x}_t, t)\|_2^2]. \quad (2.11)$$

The image-domain models of Chapter 3 use the velocity target [5]

$$\mathbf{v}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{\epsilon} - \sqrt{1 - \bar{\alpha}_t}\mathbf{x}_0, \quad (2.12)$$

with loss

$$\mathcal{L}_{\mathbf{v}} = \mathbb{E}_{\mathbf{x}_0, t, \boldsymbol{\epsilon}} [\|\mathbf{v}_t - \hat{\mathbf{v}}_{\theta}(\mathbf{x}_t, t)\|_2^2]. \quad (2.13)$$

2.3.1 Flow matching

The video models of Chapter 4 use the flow-matching view [6]. Time $t \in [0, 1]$ interpolates linearly from data to noise,

$$\mathbf{x}_t = (1 - t)\mathbf{x}_0 + t\boldsymbol{\epsilon}, \quad \mathbf{u}_t = \boldsymbol{\epsilon} - \mathbf{x}_0, \quad (2.14)$$

and the denoiser predicts the velocity field

$$\mathcal{L}_{\text{flow}} = \mathbb{E}_{\mathbf{x}_0, t, \epsilon} [\|\mathbf{u}_t - \hat{\mathbf{u}}_{\theta}(\mathbf{x}_t, t)\|_2^2]. \quad (2.15)$$

Sampling integrates the learned velocity from $t = 1$ (noise) to $t = 0$ (data). In later chapters we often write the denoising network as f_{θ} and its predicted velocity as $\mathbf{v}_{\theta}$ even when the underlying objective is flow matching.

2.3.2 Latent diffusion

Most large image and video diffusion models operate in a compressed latent space. A pretrained VAE encoder $\mathcal{E}$ maps pixels to latents and a decoder $\mathcal{D}$ maps latents back:

$$\mathbf{z} = \mathcal{E}(\mathbf{x}), \quad \mathbf{x} \approx \mathcal{D}(\mathbf{z}). \quad (2.16)$$

The diffusion process is then applied to $\mathbf{z}$ rather than pixels. For Stable Diffusion 2.1 [7], this gives an $8\times$ spatial compression and four latent channels. For the video DiT used in Chapter 4, the VAE is spatio-temporal and is described next.

2.3.3 Conditioning and classifier-free guidance

Conditional diffusion models take side information $\mathbf{c}$, such as text or a G-buffer, and learn $f_{\theta}(\mathbf{x}_t, t, \mathbf{c})$. Classifier-free guidance (CFG) [8] trains the same network with the condition sometimes dropped. At inference, two predictions are extrapolated:

$$\hat{\mathbf{v}}_{\theta}^{\text{CFG}} = \hat{\mathbf{v}}_{\theta}(\mathbf{x}_t, t, \emptyset) + w [\hat{\mathbf{v}}_{\theta}(\mathbf{x}_t, t, \mathbf{c}) - \hat{\mathbf{v}}_{\theta}(\mathbf{x}_t, t, \emptyset)], \quad (2.17)$$

with guidance weight $w \geq 1$. The same expression applies to noise prediction, velocity prediction, and flow matching by changing the predicted quantity. Chapter 3 uses condition dropout to accept arbitrary G-buffer subsets; Chapter 4 uses positive and negative text prompts to steer $X \rightarrow \text{RGB}$ toward real-looking video.

2.3.4 Fine-tuning pretrained diffusion models

Specialized diffusion models are usually obtained by fine-tuning a pretrained base. LoRA [9] freezes the base and learns low-rank updates,

$$W_{\text{LoRA}} = W + BA, \quad A \in \mathbb{R}^{r \times d_{\text{in}}}, \quad B \in \mathbb{R}^{d_{\text{out}} \times r}. \quad (2.18)$$

ControlNet [10] adds a trainable conditioning tower. Concatenation-based conditioning inserts dense conditions into the latent tensor itself. Chapter 3 uses channel-wise concatenation for image diffusion; Chapter 4 instead shows that video DiTs are better adapted by frame-wise concatenation, where different latent frame slots become different input and output modalities.

2.3.5 Diffusion transformers for video

Diffusion transformers (DiTs) [11] replace the U-Net denoiser with a transformer over latent tokens. Chapter 4 uses Wan 2.1 [12]; related video DiTs include HunyuanVideo [13] and Cosmos [14].

Video variational autoencoders. Let $\mathbf{Y}_{1:F}^m$ denote an F -frame sequence of modality m :

$$\mathbf{Y}_{1:F}^m = (\mathbf{Y}_1^m, \dots, \mathbf{Y}_F^m), \quad m \in \mathcal{M}. \quad (2.19)$$

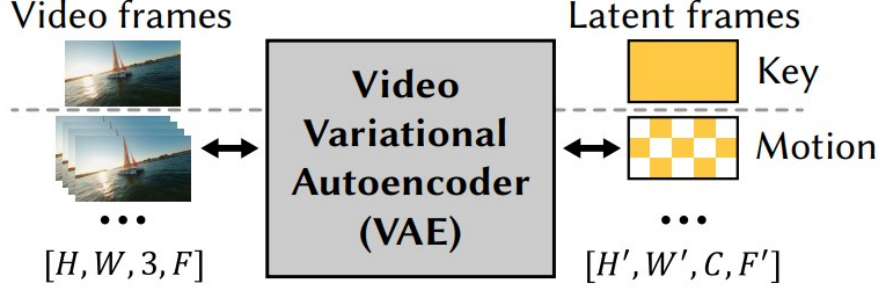


Figure 2.3: Video VAE compression used by modern text-to-video diffusion models. A key frame is encoded as a latent keyframe, and each following group of four RGB frames is encoded as one latent motion frame.

A video VAE encodes this sequence into a latent tensor

$$\mathbf{x}^m = \mathcal{E}_{\text{vae}}(\mathbf{Y}_{1:F}^m), \quad \text{shape}(\mathbf{x}^m) = H' \times W' \times C \times F'. \quad (2.20)$$

For Wan 2.1, $H' = H/8$, $W' = W/8$, and $C = 16$. Temporally, $4k + 1$ RGB frames become $k + 1$ latent frames: one latent keyframe followed by k latent motion frames. Equivalently,

$$F = 4k + 1, \quad F' = k + 1 = \frac{F + 3}{4}. \quad (2.21)$$

Patchification and tokens. A learned patchify map $\mathcal{P}$ converts each $2 \times 2 \times C$ latent block at spatial index (i, j) and latent frame k into a token:

$$\mathbf{u}_{i,j,k}^m = \mathcal{P}(\mathbf{x}_{2i:2i+1, 2j:2j+1, :, k}^m). \quad (2.22)$$

The number of tokens is

$$N_{\text{tok}} = \frac{H'}{2} \cdot \frac{W'}{2} \cdot F'. \quad (2.23)$$

Wan 2.1 uses token dimension $T = 6144$. The transformer operates on this token set; an unpatchify head maps output tokens back to latent blocks.

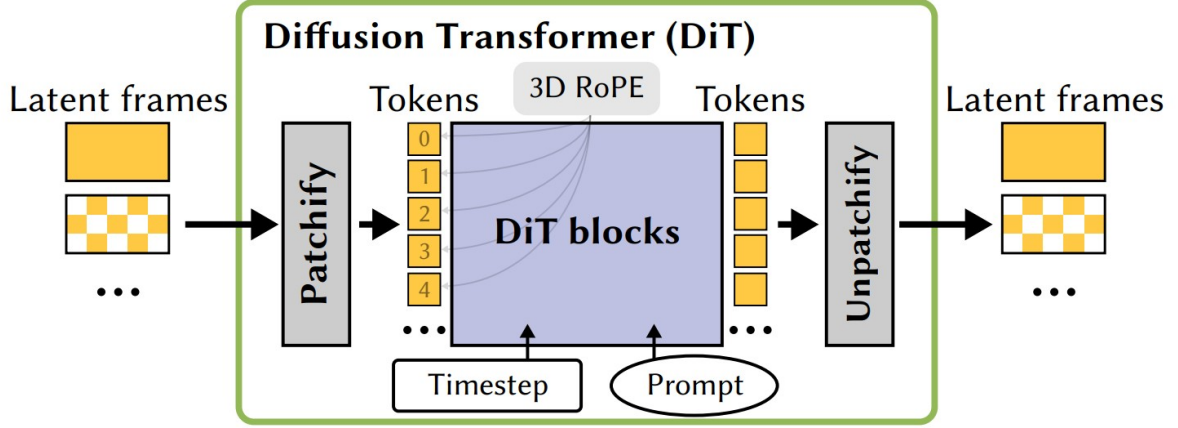


Figure 2.4: A video DiT patchifies latent frames into tokens, injects position through embeddings, and processes the resulting token set with transformer layers conditioned on text and diffusion time.

Positional and conditioning embeddings. Token order alone does not define spatial or temporal identity. A 3D rotary position embedding (RoPE) [15] rotates query and key vectors according to token index $\ell = (i, j, k)$:

$$\mathbf{Q}_{i,j,k}^m = \mathcal{R}_{i,j,k}(W_Q \mathbf{u}_{i,j,k}^m), \quad \mathbf{K}_{i,j,k}^m = \mathcal{R}_{i,j,k}(W_K \mathbf{u}_{i,j,k}^m). \quad (2.24)$$

Text prompt p is injected through cross-attention via an encoded prompt $\tau(p)$, and the diffusion time t modulates transformer blocks through adaptive layer normalization. Abstractly, the video DiT predicts

$$\mathbf{v}_\theta = f_\theta(\{\mathbf{u}_{i,j,k}\}, t, \tau(p)). \quad (2.25)$$

This token view is crucial for Chapter 4: RGB and G-buffer frames can be placed in the same token set, and learned type information can tell the model which tokens are clean inputs, noisy outputs, RGB, albedo, depth, lighting, or material.

Streaming video generation. Base video DiTs generate fixed-length clips. Wan 2.1 is trained most naturally around 81 RGB frames, corresponding to $F' = 21$ latent frames. Longer sequences require processing chunks while carrying information across chunk boundaries. Teacher forcing trains each chunk conditioned on clean previous frames, but at inference the previous frame is generated and may contain errors. Self forcing [16] trains on the model’s own predictions to reduce this train-test gap. Long-context tokens, inspired by FramePack [17], add reference-frame tokens so that each chunk sees coarse sequence-level appearance. These are the three ingredients combined in Chapter 4.

2.4 Summary

Rendering supplies the physical semantics of the G-buffer channels; inverse rendering motivates estimating those channels from real images; diffusion and video DiTs supply the learned priors. With this notation in place, Chapter 3 instantiates $\text{RGB} \leftrightarrow \text{X}$ with image latent diffusion, and Chapter 4 instantiates it with typed video-DiT tokens for images, videos, streams, and a unified forward/inverse model.

Chapter 3

Controllable Image Synthesis from G-buffers

The work described in this chapter was published as “ $RGB \leftrightarrow X$: Image decomposition and synthesis using material- and lighting-aware diffusion models” at SIGGRAPH 2024 [18].

In this chapter, we develop the first instance of $RGB \leftrightarrow X$, in the image domain. As argued in Chapter 1, we ask whether the G-buffers exposed by a classical rendering pipeline – albedo, normal, roughness, metallicity, and a lighting summary – carry enough structure to control a learned generative model, and whether they leave enough freedom for that model to synthesize the photorealism that traditional rendering pays a high price to author. The simplest setting in which to test this question is a still image, with a single pretrained backbone and a single fixed channel set: we set aside the complications of video temporal coherence, of long-context streaming, and of training on G-buffers estimated from real data, all of which we take on in the next chapter. What remains is the substantive question itself, in its simplest form.

Our main contribution is the forward direction, $X \rightarrow RGB$, which takes the G-buffers as input and synthesises a photorealistic RGB image: it is the model an artist or a downstream pipeline would use to turn a partial scene specification into a final image, and it is where we test our claim about controllable image synthesis. We pair $X \rightarrow RGB$

with an inverse model, $\text{RGB} \rightarrow \mathbf{X}$, which takes an RGB image and estimates each of the channels. $\text{RGB} \rightarrow \mathbf{X}$ is developed here not as a contribution in its own right but as an enabler: by reading the same channels off a real photograph and feeding them back into $\mathbf{X} \rightarrow \text{RGB}$, it makes editing applications such as material replacement, recolouring, and object insertion possible without ever reconstructing a 3D scene. As a side benefit, the same $\text{RGB} \rightarrow \mathbf{X}$ model also produces the G-buffer supervision that Chapter 4 will use to push forward generation toward photorealism on real video. The two directions, trained on the same backbone with the same channel definitions, sit naturally as a pair.

Two design choices follow from running both directions in a single chapter. First, both models are fine-tuned from Stable Diffusion 2.1 [7] – the same latent-diffusion base reviewed in Section 2.3.2 – so we inherit a strong image prior on a modest training budget instead of training from scratch. Second, both models share the channel definitions of Table 2.2 and the radiometric framing of Section 2.1: they manipulate the same physical quantities, only with the direction of the arrow reversed.

A central practical problem in this setting is heterogeneous training data. No public dataset gives all five channels for the same images at sufficient quality: INTERIOR-VERSE [19] provides albedo and normals but unreliable material parameters; HYPER-SIM [20] provides irradiance and normals but no material parameters; we render our own EVERMOTION, which provides reliable material data but no irradiance; and we add a real-photograph dataset IMAGEDECOMP annotated by our own $\text{RGB} \rightarrow \mathbf{X}$ model. To use all four together we (i) repurpose the text prompt of $\text{RGB} \rightarrow \mathbf{X}$ as a per-channel modality switch, so that a single network can be trained on whichever channel happens to be available in any given dataset, and (ii) train $\mathbf{X} \rightarrow \text{RGB}$ with per-channel dropout, so that it accepts any subset of the channels at inference time and creatively fills in the rest. Both ideas reuse mechanisms already present in the base diffusion model – text conditioning and the classifier-free guidance dropout of Section 2.3.3. Perhaps unexpectedly, both

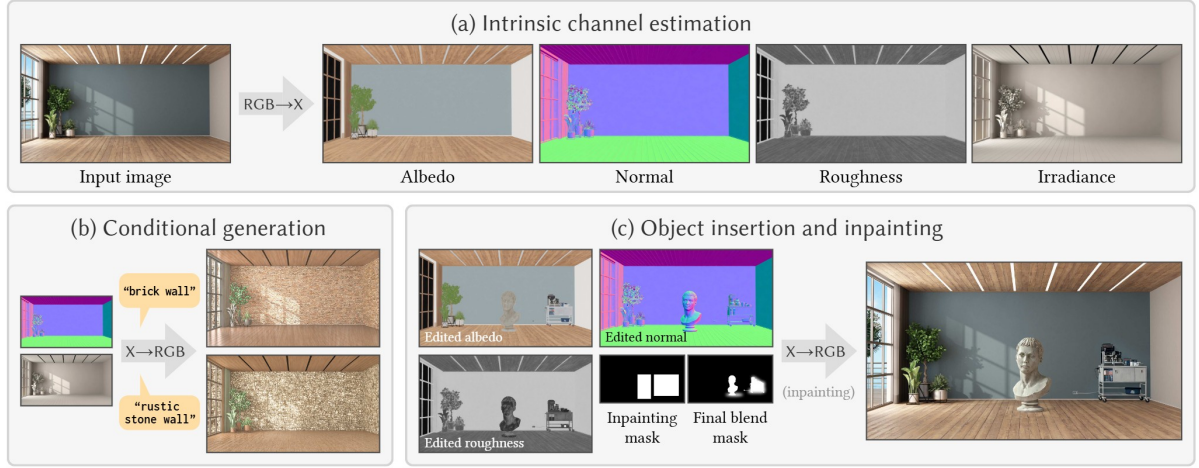


Figure 3.1: Two diffusion models, $\text{RGB} \rightarrow \text{X}$ for image decomposition into G-buffers and $\text{X} \rightarrow \text{RGB}$ for image synthesis from such G-buffers, in a unified conditional diffusion framework. (a) The $\text{RGB} \rightarrow \text{X}$ model produces clean, plausible estimates of the G-buffers X from a real input image. (b) New photorealistic images are produced by the $\text{X} \rightarrow \text{RGB}$ model, here from a subset of the estimated G-buffers plus a text prompt. (c) Synthetic objects inserted into the estimated G-buffers are composited by an inpainting version of the $\text{X} \rightarrow \text{RGB}$ model, with appropriate masks, into a final image with matching lighting and shadows.

also turn out to give the user useful control at inference, not merely a way to train on incomplete data.

The rest of this chapter is organized as follows. Section 3.1.1 fixes the channel set and notes how it specializes the general definitions of Table 2.2 for the indoor-scene image setting; Section 3.1.2 describes the four training datasets and how they cover this set. Section 3.2 introduces the $\text{RGB} \rightarrow \text{X}$ model, including the prompt-as-modality-switch trick that handles heterogeneous training data. Section 3.3 introduces the $\text{X} \rightarrow \text{RGB}$ model, including channel dropout, a low-resolution lighting hint, and an inpainting variant for local editing. Section 3.4 evaluates both models against prior work on synthetic and real images, and demonstrates editing applications – material replacement and object insertion – that the forward and inverse models enable jointly. Section 3.5 summarizes what was learned and motivates the move, in the next chapter, from images to video and from synthetic to real training data.

3.1 G-buffers and datasets

3.1.1 G-buffers in this chapter

The general G-buffer set used throughout the dissertation is collected in Table 2.2; in this chapter, we restrict attention to a fixed subset $\mathbf{X} = (\mathbf{n}, \mathbf{a}, \mathbf{r}, \mathbf{m}, \mathbf{E})$, with definitions and shapes as in that table. A few specializations to the indoor-image setting of this chapter are worth noting. The albedo $\mathbf{a}$ doubles as the diffuse base color for opaque dielectrics and the specular base color for metals, following the metallic-roughness convention of Section 2.1.3. The roughness $\mathbf{r}$ is taken as the perceptually linear parameter of [2]: high values denote matte materials, low values denote shiny ones. The metallicity $\mathbf{m}$ is a linear blend weight between the dielectric and metallic microfacet lobes. The diffuse irradiance $\mathbf{E}$ is defined exactly as in Equation (2.4), but in this chapter it is computed and consumed at the same spatial resolution as the RGB image; an alternative low-resolution treatment is discussed in Section 3.3. We omit the depth channel of Table 2.2 because depth on indoor scenes is well predicted from the normals and adds little information that the other channels do not already provide.

Unlike a full BRDF system in a production renderer, the five-channel parametrization above cannot represent every real material. Glass is the canonical exception: we treat it as fully opaque with zero roughness and metallicity. In practice, this is rarely a problem in the image-domain setting of this chapter, because both $\text{RGB} \rightarrow \mathbf{X}$ and $\mathbf{X} \rightarrow \text{RGB}$ infer from context that an object is a window or a glass cabinet, and plausibly fill in objects or illumination behind it.

Table 3.1: We combine four heterogeneous datasets (ours in bold), each providing a subset of the channels we need for training. For each dataset we mark channels as available (✓), unavailable (✗), or available but not fully reliable (✓). We also include representative images from the datasets. IMAGEDCOMP is an RGB-only dataset for which we estimated the G-buffers using our RGB→X model.

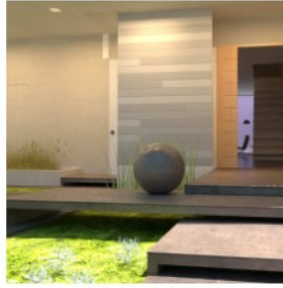
Dataset	Size	Albedo	Normal	Roughness	Metallic.	Irrad.
INTERIORVERSE	50,097	✓	✓	✓	✓	✗
HYPERSIM	73,819	✓	✓	✗	✗	✓
Evermotion	17,000	✓	✓	✓	✓	✗
ImageDecomp	50,000	✓	✓	✓	✓	✓



INTERIORVERSE



HYPERSIM



EVERMOTION



IMAGEDCOMP

3.1.2 Datasets

To train both RGB→X and X→RGB we ideally want a large, high-quality dataset of images paired with reliable channels for each pixel: **I** together with (**n**, **a**, **r**, **m**, **E**) and a text caption. No existing dataset satisfies this requirement at the necessary scale, so we instead piece together four datasets with partial coverage and annotate one further dataset with our own RGB→X model. Table 3.1 summarizes the size and per-channel coverage of all four; the per-channel modality switch in Section 3.2 and the per-channel dropout in Section 3.3 are what let us train on this heterogeneous mix.

INTERIORVERSE [19] is a synthetic indoor-scene dataset of over 50,000 rendered images, with **n**, **a**, **r**, and **m** channels alongside the rendered image **I**. Two issues required attention. The rendered images themselves contain Monte Carlo noise, which does not affect RGB→X training but causes X→RGB to learn to reproduce the noise; we there-

fore denoise all INTERIORVERSE renderings with the off-the-shelf NVIDIA OptiX denoiser [21]. The roughness and metallicity values were also often dubious, so we exclude these channels from INTERIORVERSE during training. The dataset has a recognizably synthetic look that an $X \rightarrow \text{RGB}$ model trained exclusively on it would happily imitate, and its small variety of objects and materials introduces undesirable correlations (e.g. green albedo is strongly correlated with plant geometry, so a green wall would tend to grow leaves).

HYPERSIM [20] is a second synthetic photorealistic dataset of over 70,000 rendered images, this time with $\mathbf{n}$, $\mathbf{a}$, and – most importantly for our setting – $\mathbf{E}$. It does not provide $\mathbf{r}$ or $\mathbf{m}$, and occasionally bakes specular shading into $\mathbf{a}$; the latter is rare enough that we still use the $\mathbf{a}$ data. HYPERSIM broadens the appearance variety relative to INTERIORVERSE, but is still not enough on its own for highly photorealistic synthesis.

We complete the synthetic side with our own EVERMOTION. We rendered 17,000 images of 85 indoor scenes by placing cameras randomly along pre-recorded paths through artist-made interiors, in the same style as INTERIORVERSE. The point of EVERMOTION is reliable $\mathbf{r}$ and $\mathbf{m}$, for which it is currently the only trustworthy source we have access to.

To push $X \rightarrow \text{RGB}$ toward photorealism we then add 50,000 high-quality commercial interior images from photographs and high-quality renderings, with no available G-buffers. We annotate the full set with $\mathbf{n}$, $\mathbf{a}$, $\mathbf{r}$, $\mathbf{m}$, and $\mathbf{E}$ using our own $\text{RGB} \rightarrow X$ model. The resulting RGB+G-buffer dataset is what we call IMAGEDECOMP. To preserve the text-understanding ability of the base diffusion model during fine-tuning, we additionally compute BLIP-2 [22] captions for every image in all four datasets.

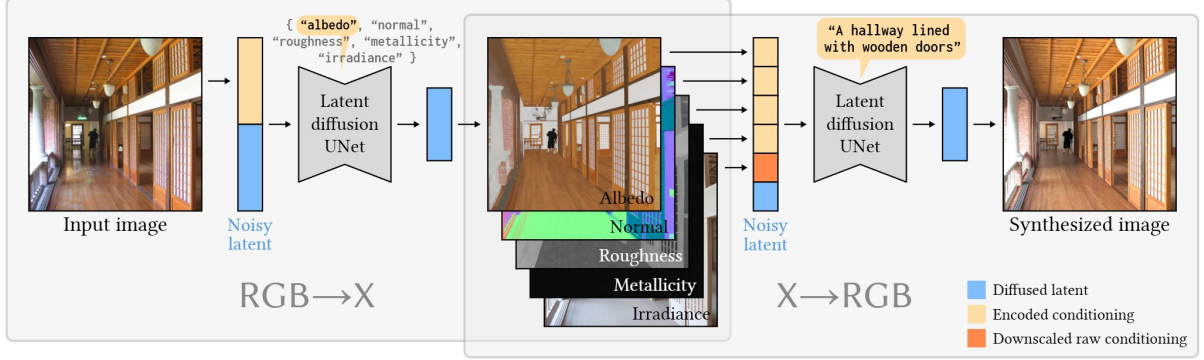


Figure 3.2: High-level overview of our two diffusion models. **Left:** The RGB→X model takes the input image, encoded into latent space by the pre-trained encoder, concatenated with the diffusion latent. We repurpose the text prompt to a switch choosing the desired output channel; this allows training with datasets containing any subset of the supported channels. **Right:** The X→RGB model concatenates the input G-buffers, again encoded by the pre-trained encoder, with the diffusion latent. One exception is the irradiance (lighting) channel, which is downsampled to latent resolution rather than passed through the encoder. This model can accept usual text prompts. All input conditions to X→RGB are optional.

3.2 The RGB→X model

In this section, we describe our RGB→X model to estimate the G-buffers $\mathbf{X}$ from an input RGB image $\mathbf{I}$. The output contains all channels discussed in Section 3.1.1. Similarly to [23], we fine-tune a pre-trained text-to-image latent diffusion model, Stable Diffusion 2.1 [7]; we recap the relevant base-model components in Sections 2.3.2 and 2.3.3. Figure 3.2 shows a high-level overview of our model.

Like Stable Diffusion, our RGB→X model operates on a latent space with a pre-trained encoder $\mathcal{E}$ and decoder $\mathcal{D}$. During inference, it takes $\mathcal{E}(\mathbf{I})$ as input condition and iteratively denoises a Gaussian-noise latent image $\mathbf{z}_T^{\mathbf{X}}$ to produce the target latent image $\mathbf{z}_0^{\mathbf{X}}$ encoding the G-buffers $\mathbf{X}$. During training, we optimize the velocity-prediction loss of Equation (2.13), which we found to outperform the equivalent ϵ -prediction loss in this setting:

$$\mathbf{v}_t^{\text{RGB} \rightarrow \mathbf{X}} = \sqrt{\bar{\alpha}_t} \epsilon - \sqrt{1 - \bar{\alpha}_t} \mathbf{z}_0^{\mathbf{X}}, \quad (3.1)$$

$$L_{\theta} = \left\| \mathbf{v}_t^{\text{RGB} \rightarrow \text{X}} - \hat{\mathbf{v}}_{\theta}^{\text{RGB} \rightarrow \text{X}}(t, \mathbf{z}_t^{\text{X}} \mathcal{E}(\mathbf{I}), \tau(\mathbf{p}^{\text{X}})) \right\|_2^2. \quad (3.2)$$

Here t is the noise amount (“time step”) drawn uniformly during training, $\epsilon \sim \mathcal{N}(0, 1)$, $\bar{\alpha}_t$ is a scalar function of t , $\hat{\mathbf{v}}_{\theta}^{\text{RGB} \rightarrow \text{X}}$ is our RGB→X diffusion model with parameters θ , $\mathbf{z}_0^{\text{X}}$ is the target latent, $\mathbf{z}_t^{\text{X}}$ is the noisy latent after adding noise ϵ at time step t to $\mathbf{z}_0^{\text{X}}$. Further, $\mathbf{p}^{\text{X}}$ is the text prompt computed for $\mathbf{I}$, and τ is the CLIP text encoder [24] that encodes the prompt into a text-embedding vector. This CLIP embedding is used as a specialized context for cross-attention layers of the model.

Image and G-buffer encoding As our model operates in latent space, we encode the input image as $\mathcal{E}(\mathbf{I})$ and concatenate it to the noisy latent $\mathbf{z}_t^{\text{X}}$ as input to our $\hat{\mathbf{v}}_{\theta}^{\text{RGB} \rightarrow \text{X}}$ model. We use the frozen encoder $\mathcal{E}$ from the original Stable Diffusion model, which we also found to work well for all our G-buffer encoding.

Handling multiple output channels The output of the original Stable Diffusion model is a 4-channel latent image which can be decoded into a single RGB image. As we aim to produce additional output channels (albedo $\mathbf{a}$, normal $\mathbf{n}$, roughness $\mathbf{r}$, metallicity $\mathbf{m}$, and lighting $\mathbf{E}$) we may expect that a larger latent vector would help better encode the information, as done in previous work [23]. However, we find that extending the number of latent channels of the original model leads to lower-quality results. Adding more latent channels to the operating latent space of a diffusion model forces us to re-train both input and output convolutional layers from scratch. The model is suddenly “shocked” into a new domain, making the training more challenging.

We train our model with various datasets to increase variety, as described in Section 3.1.2, but this comes with the additional issue of heterogeneous G-buffer coverage across datasets, which is challenging for an approach that stacks all G-buffer channels into a larger latent. A straightforward approach would be to compute the loss only for the

channels that are available in each training iteration. We however found this approach to perform poorly.

Our solution is to produce a single G-buffer channel at a time and repurpose the input text prompt (which does not serve any other purpose in our $\text{RGB} \rightarrow \text{X}$ task) as a “switch” to control the diffusion-model output. Previous work [25, 26] shows that it is possible to use specially designed prompts as instructions to control a single diffusion model to perform different tasks explicitly. Inspired by this design, we use five fixed prompts acting as switches. More specifically, one unit of data is collated as $\{\mathbf{g}, \mathbf{p}^{\text{X}}, \mathbf{I}\}$ where $\mathbf{g} \in \{\mathbf{n}, \mathbf{a}, \mathbf{r}, \mathbf{m}, \mathbf{E}\}$ and $\mathbf{p}^{\text{X}} \in \{\text{‘normal’}, \text{‘albedo’}, \text{‘roughness’}, \text{‘metallicity’}, \text{‘irradiance’}\}$ is set accordingly. We find that this approach performs similarly to fine-tuning separate models for each output modality in $\{\mathbf{n}, \mathbf{a}, \mathbf{r}, \mathbf{m}, \mathbf{E}\}$ while fine-tuning and storing only a single network’s weights.

3.3 The $\text{X} \rightarrow \text{RGB}$ model

We now describe our $\text{X} \rightarrow \text{RGB}$ model, performing photorealistic RGB image synthesis from G-buffers X , illustrated in Figure 3.2. As with $\text{RGB} \rightarrow \text{X}$, we fine-tune a diffusion model starting from Stable Diffusion 2.1, with several different considerations.

In the $\text{X} \rightarrow \text{RGB}$ case, we define the target latent variable as $\mathbf{z}_0^{\text{RGB}} = \mathcal{E}(\mathbf{I})$, directly encoding the image $\mathbf{I}$. We provide the input condition X through concatenation of the encoded input G-buffers, adjusted to take various dataset properties into consideration as described below. When using all G-buffers, the input latent vector is defined as

$$\mathbf{z}_t^{\text{X}} = (\mathcal{E}(\mathbf{n}), \mathcal{E}(\mathbf{a}), \mathcal{E}(\mathbf{r}), \mathcal{E}(\mathbf{m}), \mathcal{E}(\mathbf{E})). \quad (3.3)$$

We train our $X \rightarrow RGB$ model by minimizing the loss function L'_θ :

$$\mathbf{v}_t^{X \rightarrow RGB} = \sqrt{\bar{\alpha}_t} \epsilon - \sqrt{1 - \bar{\alpha}_t} \mathbf{z}_0^{RGB}, \quad (3.4)$$

$$L'_\theta = \left\| \mathbf{v}_t^{X \rightarrow RGB} - \hat{\mathbf{v}}_\theta^{X \rightarrow RGB} \left(t, \mathbf{z}_t^{RGB}, \mathbf{z}_t^X, \tau(\mathbf{p}) \right) \right\|_2^2. \quad (3.5)$$

Here $\hat{\mathbf{v}}_\theta^{X \rightarrow RGB}$ is our $X \rightarrow RGB$ diffusion model with parameters θ . We concatenate the noisy latent $\mathbf{z}_t^{RGB}$ and the conditioning latent $\mathbf{z}_t^X$ together before feeding it into $\hat{\mathbf{v}}_\theta^{X \rightarrow RGB}$. The CLIP text embedding $\tau(\mathbf{p})$ is used as the context for cross-attention layers of the model. Unlike $RGB \rightarrow X$, the text prompt here carries actual scene information and serves as an additional control, as in standard text-to-image diffusion.

While our $RGB \rightarrow X$ model required a solution to output multiple modalities, the $X \rightarrow RGB$ model only requires changing the input layers to handle the additional conditional latent channels. As in the original Stable Diffusion, the output remains a single RGB image. During training, only the newly added weights of the input convolutional layer need to be trained from scratch to handle the additional conditions, which does not “shock” the model out of its normal denoising ability for $\mathbf{z}_t^{RGB}$.

Handling heterogeneous data The problem of different G-buffer channels missing from different datasets remains. To resolve this issue, we follow the observations of previous work [8, 27], which propose to jointly train a conditional and unconditional diffusion model through condition-channel dropout to improve sample quality and enable image generation with any subset of conditions; the same per-condition dropout mechanism underlies classifier-free guidance (Section 2.3.3). We therefore use a G-buffer dropout strategy; with it, our conditioning latent $\mathbf{z}_t^X$ can be rewritten as

$$\mathcal{P}(x) \in \{\mathcal{E}(x), 0\} \quad (3.6)$$

$$\mathbf{z}_t^X = (\mathcal{P}(\mathbf{n}), \mathcal{P}(\mathbf{a}), \mathcal{P}(\mathbf{r}), \mathcal{P}(\mathbf{m}), \mathcal{P}(\mathbf{E})). \quad (3.7)$$

This approach lets us handle heterogeneous datasets during training, and lets us choose which inputs to provide at inference; for example, providing no albedo or no lighting will result in the model generating plausible images, using its prior to compensate for the missing information (Figure 3.6).

Low-resolution lighting Our RGB→X model succeeds in estimating highly detailed lighting in the form of a diffuse irradiance image $\mathbf{E}$, closely following high-resolution geometry and normals. While this could be beneficial for some applications, using these detailed lighting buffers for X→RGB presents an issue if we want to edit the detailed normals and control the lighting using a coarser interpretation of $\mathbf{E}$. We would like to provide the lighting as a “hint” to the X→RGB model rather than a precise per-pixel control. Instead of encoding the full-resolution lighting $\mathbf{E}$ into the latent space as for the other conditions, we simply downsample it into the same resolution as the latent. Doing so provides the X→RGB model with a coarser hint of lighting without pixel detail, while still achieving adherence to the overall lighting condition. This matters, for example, when editing the normals in Figure 3.7.

Fine-tuning for inpainting. To enable local editing applications (shown in Figures 3.1 and 3.7) we fine-tune our X→RGB model to support inpainting by simply adding a masked image and a mask channel to the model input. We downsample the mask to the latent-space resolution and concatenate it to the conditioning latent $\mathbf{z}_t^X$.

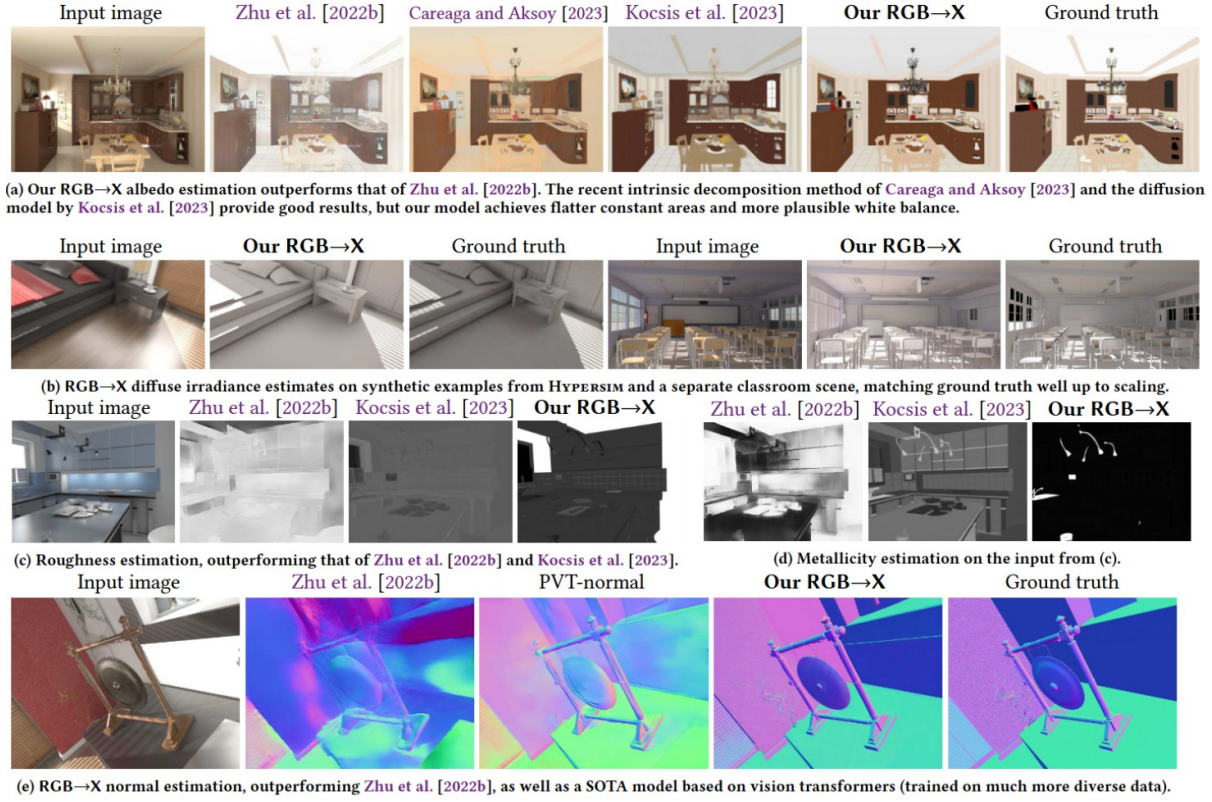


Figure 3.3: **Synthetic data comparison** of our RGB→X model against previous methods [19, 28] and a known ground truth. All input images and ground truths are from HYPERSIM, except for the classroom scene (c).

3.4 Results

Note about picking results from a generative model Applying generative models to the RGB→X and X→RGB problems means that the output is not unique but sampled from a distribution. While one could evaluate a number of samples and take their mean [23], we do not recommend this approach, as it can blur details that have been reasonably estimated within each sample. Instead, we pick a single sample to display in each comparison. Albedo, lighting, and normal samples are typically usable on the first try, while roughness and metallicity require more sample selection due to the lack of reliable training data and the inherent ambiguity of these properties.

3.4.1 RGB→X on synthetic and real inputs

Figures 3.3 and 3.4 show our results on G-buffer estimation for synthetic and real examples. None of the synthetic input examples were part of the training data.

Albedo We compare albedo estimation against previous work in Figure 3.3(a) for synthetic and Figure 3.4(a, b) for real inputs. Generally, our model is best at removing reflections, highlights, shadows, and color cast from the inputs, while providing the flat-test estimates for albedo regions that should indeed be constant. The method of [19] performs worse on both synthetic and real inputs, hinting at the limitations of non-generative models, nor does it incorporate any special design for the albedo estimation problem. The recent intrinsic decomposition method of [28] provides good results, but our model achieves flatter constant areas and a more plausible white balance. The same holds when comparing with the diffusion model by [23], even though that model also produces strong results. For example, on the bedroom photo in Figure 3.4(a, top row), our model is the only one correctly predicting that all bed-linen pixels should have identical white albedo. The challenging real image in Figure 3.4(b) also produces a clean albedo estimate that outperforms other methods, though our model removes some wear from the wooden floor, possibly due to training on synthetic materials without wear.

Diffuse irradiance (lighting) In Figure 3.3(b), we see that our model produces diffuse irradiance estimates that closely match ground truth on synthetic data, even on inputs with intricate shadow patterns, and with very little to no leaking of material properties into the estimation. The color in the irradiance is also plausibly shifted away from pure white under colored lighting. Our estimates are also plausible on real inputs, such as in Figure 3.4(b). [28] do not provide irradiance directly, so we divide the original image by their predicted albedo and use the resulting approximate irradiance as a baseline.

Metallicity and roughness As shown in Figure 3.3(c,d) and Figure 3.4(c,d), our RGB→X model generates much more plausible roughness and metallicity for a given input image than the previous publicly available state of the art [19, 23]. These material properties are challenging to recover accurately, for two reasons. First, the amount of reliable training data for them is the lowest. Second, they only impact surface reflection significantly if lit by appropriate high-frequency illumination; otherwise the model has to revert to prior knowledge, estimating what the object could be and whether such objects tend to be rough or metallic. These issues translate to a higher sampling variance of our model, and a lower yield of “good” samples.

Normals In synthetic tests (Figure 3.3(e)) as well as real ones (Figure 3.4(e)), our model estimates normals plausibly, including high-frequency geometry, while correctly predicting flat normals for flat surfaces even if they have texture or high-frequency lighting. Our results outperform those of [19] and slightly improve on the state-of-the-art PVT-normal, an internal Pyramid Vision Transformer [29] baseline trained on the same diverse data as MiDaS [30, 31]. We do not claim general improvement in normal estimation, as PVT-normal is specifically designed to work well across general images; we provide this comparison for the sake of completeness.

Quantitative comparisons For albedo, normal, roughness and metallicity estimation, we compare to the corresponding previous methods in Table 3.2. Our RGB→X has the best PSNR and LPIPS values on all channels, with the exception of irradiance for which we do not have existing methods to compare to.

3.4.2 $X \rightarrow \text{RGB}$ model results

Comparison to path tracing reference In Figure 3.5, we validate that our $X \rightarrow \text{RGB}$ model produces results closely matching traditional Monte Carlo path tracing, as long as the input channels X are not far from the training distribution of synthetic interiors. Here, we use a common synthetic kitchen scene that is not part of our training data. We use all G-buffers (shown on the left) and feed them into our model, along with a text prompt. The result matches the path-traced reference well in terms of material appearance and global illumination. Differences can also be noted: for example, the stove has a dark metallic material in the input channels, which is rare in the training data; our model generates a brighter aluminum material, matching the metallicity instead of the albedo channel.

Subsets of input channels and text prompts. Figure 3.6 demonstrates the ability of our $X \rightarrow \text{RGB}$ model to generate plausible images by specifying only a subset of the appearance properties as input. Furthermore, text prompts can be used for additional control. Here, we control the lighting (a) or object colors (b). Generally, text control works well only when there are few objects (e.g., one sofa and a few pillows). It is hard to control the color of a specific object by text, but this issue is a common challenge for all diffusion models.

3.4.3 Applications

Material replacement In the top-left example of Figure 3.7, we edit the normal and albedo of the sofa (estimated by $\text{RGB} \rightarrow X$) and re-synthesize the image with our inpainting $X \rightarrow \text{RGB}$ model, resulting in a fuzzier, bumpier red couch. On the top right, we apply G-buffer estimation to the classic Cornell box image and edit the right-wall albedo to

Table 3.2: Numerical evaluation of $\text{RGB} \rightarrow \mathbf{X}$ against existing methods. Albedo, normal, and irradiance evaluations are conducted on the HYPERSIM test set. Roughness and metallicity are evaluated on the EVERMOTION test set.

Method	Albedo		Method	Normal	
	PSNR $\uparrow$	LPIPS $\downarrow$		PSNR $\uparrow$	LPIPS $\downarrow$
Our $\text{RGB} \rightarrow \mathbf{X}$	17.4	0.18	Our $\text{RGB} \rightarrow \mathbf{X}$	19.8	0.18
[19]	11.7	0.54	[19]	16.5	0.45
[28]	13.5	0.34	PVT-normal	18.8	0.30
[23]	12.1	0.41			

Method	Roughness		Metallicity		Irradiance	
	PSNR $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	LPIPS $\downarrow$
Our $\text{RGB} \rightarrow \mathbf{X}$	11.2	0.52	12.1	0.44	14.1	0.22
[19]	4.4	0.77	2.22	0.82	N/A	N/A
[23]	10.3	0.57	8.63	0.75	N/A	N/A

blue. The color bleeding in the rightmost box is correctly updated. The inpainting mask here includes a larger region, allowing for the color-bleeding correction. In the bottom example, we change the normal and albedo of the original room to edit the floor appearance to a wood floor.

Object insertion In Figure 3.1(c), we use our framework to insert new synthetic objects into an RGB image. We render the G-buffers of the new objects and composite them into the estimated G-buffers. We use our inpainting $\mathbf{X} \rightarrow \text{RGB}$ model with rectangular masks to produce a composite with correct lighting and shadows, which we finally blend with the original image using a tighter mask. The statue and coffee cart integrate well into the scene.

3.5 Summary

Our image-domain experiments answer the opening question of this chapter affirmatively, in the simplest setting we could pose it. G-buffers carry enough structure to drive a generative renderer toward a particular albedo, geometry, material, and lighting; they

leave enough freedom for that renderer to plausibly fill in the unspecified high-frequency detail. Going in the inverse direction, the same channels can be read off real photographs reliably enough to support useful editing operations – material replacement, recolouring, and object insertion – without ever reconstructing a 3D scene. The two directions share a backbone and a channel set, and are trained and evaluated together throughout the chapter.

The same experiments also expose the limitations that motivate the next chapter. Our models here operate on still images only, so temporal coherence on video is out of scope. Our training data is dominated by indoor synthetic renderings; $X \rightarrow RGB$ inherits this bias and consequently produces images with a recognizably synthetic look, even though we use the inverse model to annotate a single dataset of real images. Our lighting representation is a single diffuse irradiance buffer, with no separate handle for direct illumination, environment lighting, or relighting from arbitrary light sources. And the backbone, Stable Diffusion 2.1, predates the diffusion-transformer video models on which the field has since converged.

In Chapter 4, we address these limitations directly. We move from a U-Net image diffusion model to a pretrained diffusion transformer video backbone, extend the framework from images to short video segments and to long streaming sequences, replace the synthetic-leaning training data with G-buffers estimated from a large collection of real videos, and broaden the lighting interface to include direct illumination, partial G-buffer dropout, and blurred guidance. Together with the present chapter, this completes our argument that classical rendering’s structured controls and a generative model’s learned photorealism can be combined into a single $RGB \leftrightarrow X$ framework that scales from a single still image up to interactive video.

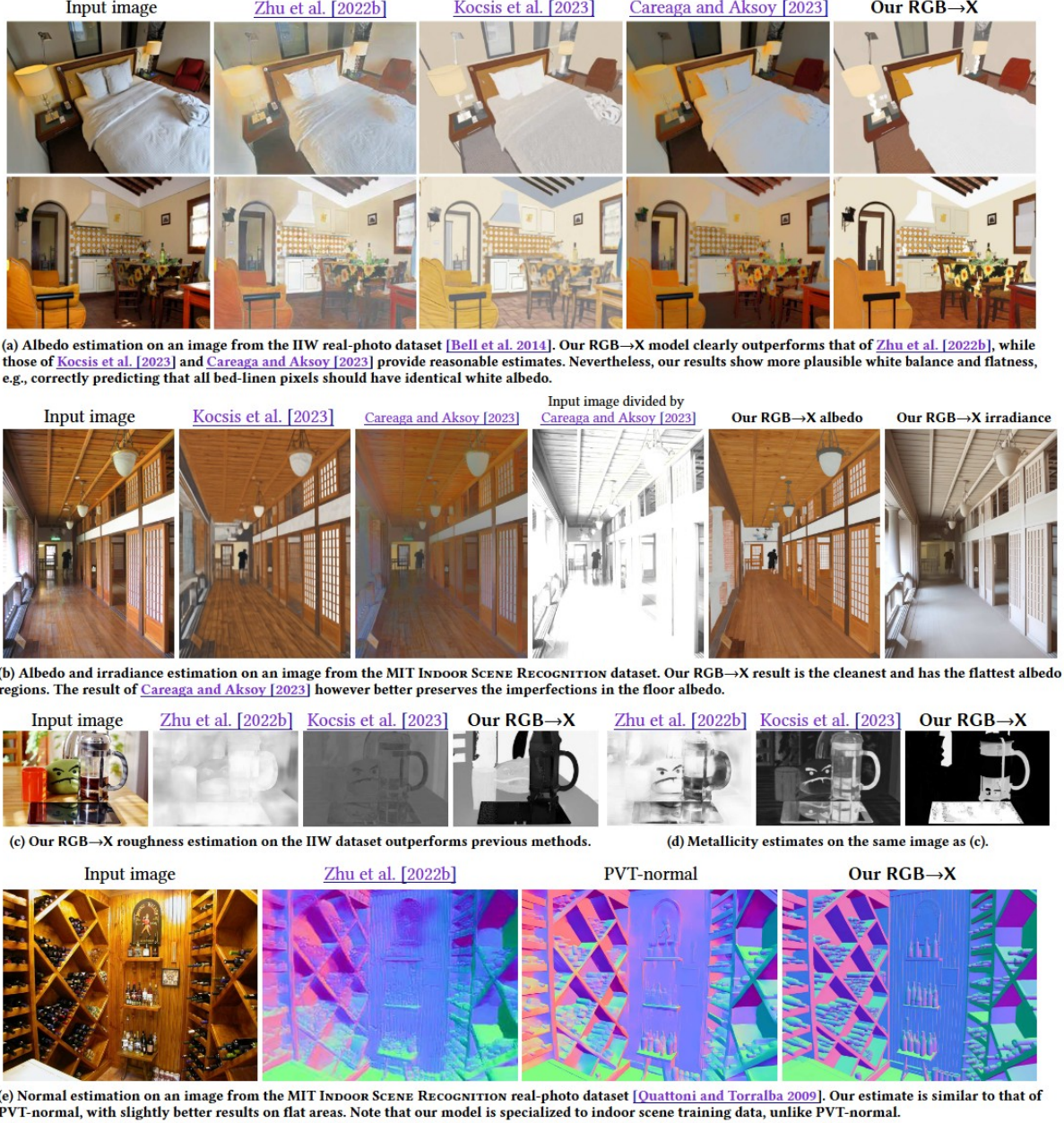
Figure 3.4: **Real-data comparison** of our RGB→X model to previous methods.



Figure 3.5: Our $X \rightarrow \text{RGB}$ result on the synthetic kitchen scene [32] which is not part of our training data. We rendered all G-buffers, shown on the left, and fed them into the model, along with a text prompt. The result matches the path-traced reference well. There are some differences, e.g., $X \rightarrow \text{RGB}$ makes the stove brighter than the requested albedo, likely because dark metallic materials are rare in the training data.



Figure 3.6: $X \rightarrow \text{RGB}$ synthesis given normal and albedo channels only, demonstrating lighting and color-control use of text prompts. (a) Starting from normal and albedo only, we show that the lighting can be controlled by text prompts to some extent. (b) Starting from normal and albedo only, we similarly show that the color of objects can be controlled by text prompts to some extent.

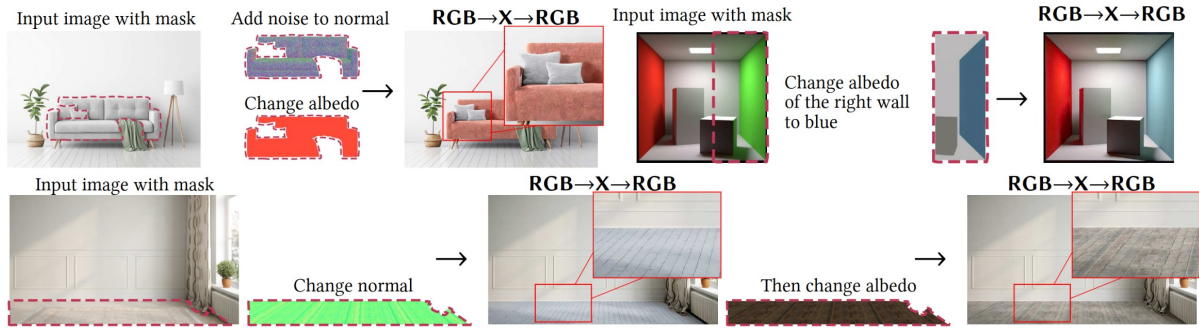


Figure 3.7: RGB→X and X→RGB models used in combination for material replacement. We show three edit examples. **Top left:** We change both normal and albedo, resulting in a fuzzier, bumpier red couch. **Top right:** We edit the right wall albedo of the Cornell box to blue and show that the colour bleeding in the rightmost box is correctly updated. **Bottom:** We first change the normal to introduce wood-plank geometry instead of the original carpet, and then also add wood albedo to edit the floor appearance.

Chapter 4

Learning Photorealistic Generative Rendering from Real Data

The work described in this chapter is currently under submission as "RGBX-Next: Towards Realistic Generative Rendering from G-Buffers."

In this chapter, we take up the photorealism gap left open at the end of Chapter 3. Our image-domain models there validated G-buffers as a control interface for generative rendering, but their training data was dominated by indoor synthetic renderings, and the resulting $X \rightarrow \text{RGB}$ model produced images with a recognizably synthetic look. We ask whether that gap can be closed – whether a generative renderer can be made indistinguishable from real photographs and videos – without the authoring cost that classical rendering incurs for every additional pixel of photorealism. We answer by learning from real data, and name the resulting framework RGBX-Next.

Our major contribution is a recipe for training $X \rightarrow \text{RGB}$ directly on real data, through a three-step closed loop. First, we train a $\text{RGB} \rightarrow X$ model – developed in this chapter as a tool rather than as a contribution in its own right – and run it on roughly six thousand real Pexels videos to produce the G-buffer annotations that no existing dataset supplies. Second, we caption every clip with a vision–language model so that real footage and the small amount of synthetic footage in the training set end up described differ-

ently. Third, we train $X \rightarrow \text{RGB}$ on the resulting G-buffer–RGB pairs, and at inference apply classifier-free guidance (Section 2.3.3) with positive “real” and negative “synthetic” prompts, biasing $X \rightarrow \text{RGB}$ away from the residual synthetic mode of its training distribution and toward the photorealistic one. The obstacle that this loop clears is that real video datasets carry no G-buffer annotations, and rendering enough real-quality G-buffer–RGB pairs with a classical pipeline would reinstate the very authoring cost that generative rendering is designed to avoid. Photorealism is therefore gained by training the generative renderer on real data through this loop, not by making any module in the pipeline more physically accurate.

We develop three further contributions to support this central one. The first is a general recipe for adapting a pretrained diffusion transformer (DiT) video backbone [11] – here Wan 2.1 [12] – into both forward and inverse generative renderers without modifying its architecture. The recipe rests on a property of DiT video models reviewed in Section 2.3.5: their core self-attention layers treat latent frames as an unordered set of tokens, and new token types can therefore be inserted into the set without altering the network. We use this property in every model in this chapter. G-buffer frames enter the token set alongside RGB frames, input frames are marked clean and denied loss while output frames are denoised as usual, and a small set of token-level mechanisms – QK type embedding, clean input tokens, and X-patchify – carry the information that was previously hard-coded into the input layer or the text prompt. We call this overall strategy frame-wise concatenation, and argue, by ablation against channel-wise concatenation and against the control-token mechanism of Wan VACE [33], that it converges faster and generalizes more readily across the many forward and inverse tasks we target in this chapter. The same recipe is what makes the $\text{RGB} \rightarrow X$ annotator strong enough to power the photorealism loop above.

The second supporting contribution broadens the lighting interface. Chapter 3 ex-

posed lighting through a single diffuse irradiance channel $\mathbf{E}$, at the same resolution as the RGB image. Here we keep $\mathbf{E}$ but add an albedo-free direct lighting channel $\mathbf{d}$ that is orthogonal to the base color; we train the model to accept varying amounts of Gaussian blur applied to the lighting buffer and to tolerate it being dropped out during part of the diffusion trajectory; and we allow partial spatial dropout of arbitrary G-buffer channels, so that an artist can specify albedo on a single segment, or normals only for part of the frame, and let the model fill in the rest. The general principle is that precision of specification is itself a parameter to tune: overly precise G-buffers constrain the generative model’s photorealism, and the appropriate balance is usually somewhere between fully specified and fully absent.

The third supporting contribution lifts the framework from fixed-length segments to arbitrarily long streams. A short-clip model on Wan 2.1, even at its largest trainable context length, is bounded by the pretrained clip length of the backbone; any truly streaming use of generative rendering needs to process video chunk by chunk while remaining coherent across chunks. We combine three of the techniques reviewed in Section 2.3.5 into a single training recipe: a teacher-forcing warm-up with mild stitching-frame augmentations, a self-forcing phase [16] that backpropagates through the model’s own predictions and eliminates most of the drift, and a small set of long-context reference tokens [17] that let each chunk see a coarse summary of the whole sequence. Applied to both $\text{RGB} \rightarrow \mathbf{X}$ and $\mathbf{X} \rightarrow \text{RGB}$, the same recipe produces streaming variants that run stably over hundreds of frames.

The rest of this chapter is organized as follows. Section 4.1 fixes the G-buffer set used throughout the chapter and states the overall frame-wise concatenation recipe; the high-level organization is summarized in Figure 4.2. Section 4.2.1 develops a baseline image and video $\text{RGB} \rightarrow \mathbf{X}$ model that uses frame-wise concatenation in its simplest form, with frame ID for input/output bookkeeping and the text prompt as an output-modality

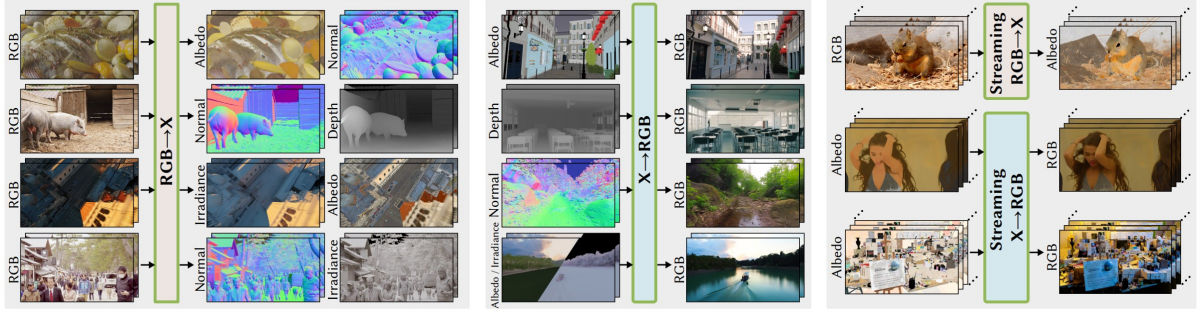


Figure 4.1: We present RGBX-Next, a unified generative framework for forward and inverse rendering, which allows estimating G-buffers from images, videos, and streams, and rendering realistic images, videos, and streams from G-buffers. **Left:** Our RGB $\rightarrow$ X model estimates clean G-buffers from 17-frame real video segments. **Middle:** Various X $\rightarrow$ RGB models trained on real RGB data produce highly realistic results from synthetically rendered or estimated G-buffers. **Right:** Streaming models finetuned from our video models produce coherent results over hundreds of frames, for RGB $\rightarrow$ X as well as X $\rightarrow$ RGB.

switch. Section 4.2.2 introduces the QK type embedding and clean input tokens, ablates them against channel-wise concatenation and Wan VACE, and compares the resulting G-buffer estimates against prior RGB $\leftrightarrow$ X models on real video; this completes the annotator on which the central contribution depends. Section 4.3 uses it to annotate real Pexels video, introduces the X-patchify module for packing multiple G-buffer inputs into a single token set, and trains X $\rightarrow$ RGB models with real/synthetic classifier-free guidance – the photorealism loop itself. Sections 4.3.3 and 4.3.4 develops lighting control through diffuse irradiance, albedo-free direct lighting, and partial dropout and blurring. Section 4.4 turns all of these models into streaming ones via teacher forcing, self forcing, and long-context tokens. The full set of resulting capabilities is illustrated in Figure 4.1.

4.1 Repurposing DiT video models for our tasks

Our main goal is to achieve controllable, realistic generative rendering: given explicit rendered G-buffers, X $\rightarrow$ RGB models should synthesize realistic RGB frames. To train

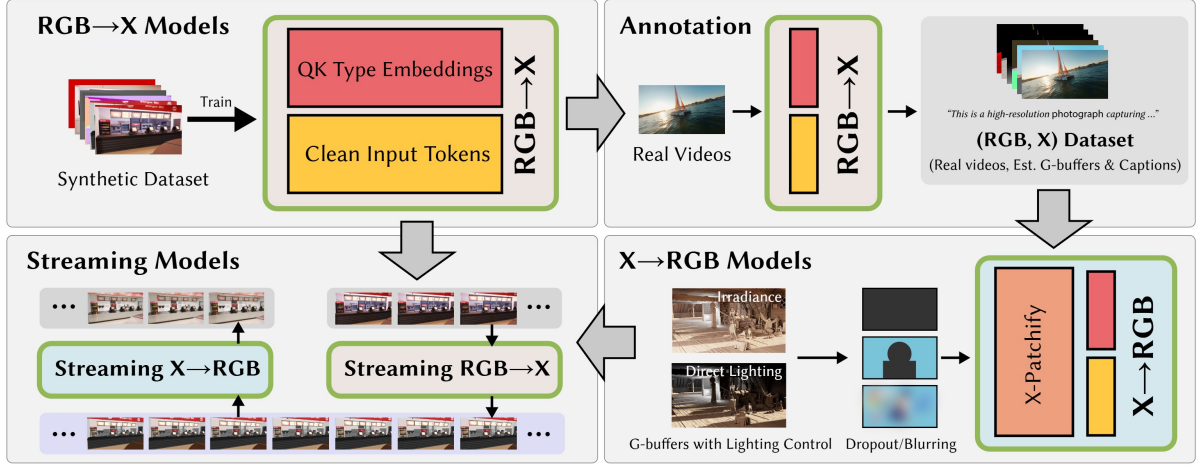


Figure 4.2: High-level overview of our chapter. Section 4.2.1 introduces a baseline video $\text{RGB} \rightarrow \text{X}$ model built on frame-wise concatenation; Section 4.2.2 strengthens it with QK type embedding and clean input tokens; Section 4.3 uses it to annotate real video and trains $\text{X} \rightarrow \text{RGB}$ models with X-patchify and real/synthetic CFG; Sections 4.3.3 and 4.3.4 adds lighting control and G-buffer dropout; Section 4.4 lifts every model to streaming.

such models on real data, we also need $\text{RGB} \rightarrow \text{X}$ models to recover G-buffers from real video frames. Across these tasks, RGB and different G-buffer channels may appear as either clean conditioning signals or generated outputs, and the available conditions may vary from one application to another. To achieve this, we develop a general *recipe* for turning text-to-video diffusion transformer (DiT) models into multi-modality, multi-input, multi-output models.

A central observation is that video DiTs operate on latent-frame sequences. This provides a natural conditioning mechanism: a subset of frame slots can encode clean inputs, while the remaining slots represent noisy outputs, with token-type information specifying each token’s role and modality. The relevant mechanics of flow matching, video VAEs, patchified DiT tokens, RoPE, cross-attention, and timestep conditioning are reviewed in Sections 2.3.1 and 2.3.5. We therefore start from that background and formalize the frame-repurposing mechanism for conditional generation in Section 4.1.1. We provide a summary of the commonly used symbols throughout this chapter in Table 4.1.

Symbol	Explanation	def.
$m, \mathcal{M}$	Modality index and modality set.	Eq. (4.1)
$\mathcal{M}_{\text{in}}, \mathcal{M}_{\text{out}}$	Input and output modality sets.	Eq. (4.1)
$\mathbf{I}, \mathbf{a}, \mathbf{n}, \mathbf{z}, \mathbf{m}, \mathbf{E}, \mathbf{d}$	RGB and G-buffer modalities.	Eq. (4.1)
$\mathbf{x}^m, \mathbf{x}_t^m, \epsilon$	Clean/noisy latent for modality m , and Gaussian noise.	Eq. (2.20), Eq. (2.14)
$\mathbf{v}_\theta, \mathcal{L}_{\text{FM}}$	Predicted flow velocity and flow-matching loss.	Eq. (2.25), Eq. (2.15)
$\mathcal{P}, \mathcal{P}_X$	Patchify and X -patchify maps.	Eq. (2.22), Eq. (4.10)
$\ell = (i, j, k), \mathbf{u}_\ell^m$	Token index and token.	Eq. (2.22)
$\mathcal{U}$	Internal token layout after patchification.	Eq. (4.5), Eq. (4.18)
$\tau = (r, m)$	Token type: role r and modality m .	Section 4.2.2
$\mathbf{e}_Q^\tau, \mathbf{e}_K^\tau$	Learned additive query/key type embeddings.	Eq. (4.8)
$\tilde{t}_\ell^\tau$	Token-specific timestep for clean input tokens.	Eq. (4.9)

Table 4.1: Core notation used in this chapter.

4.1.1 Repurposing DiT frames

In our desired forward and inverse rendering tasks, we need to support multiple modalities, some of which act as inputs and some as outputs. In forward rendering, the model takes one or more G-buffer frames as input and produces RGB frames $\mathbf{I}$ as output. In inverse rendering, this process is reversed: the input is $\mathbf{I}$, and the output is one or more corresponding G-buffers. Throughout this chapter, RGB denotes final color images or videos, and X denotes the corresponding G-buffer modalities. Thus, our modality set $\mathcal{M}$ contains RGB together with the G-buffer channels that our models may consume or generate:

$$\mathcal{M}_{\text{in}}, \mathcal{M}_{\text{out}} \subseteq \mathcal{M}. \quad (4.1)$$

The subsets $\mathcal{M}_{\text{in}}$ and $\mathcal{M}_{\text{out}}$ specify which modalities are provided as clean inputs and which modalities are outputs.

Given the input and output sets above, we place the corresponding latents into different latent frame slots of the video DiT. A text-to-video DiT model is originally designed to take noisy latent frames and produce clean latent frames, which after a number of denoising iterations become the final output of the model. We repurpose this frame in-

terface by treating some frame slots as inputs: output frames have noise added and are denoised, while input frames are provided without noise and the corresponding tokens can be discarded after processing by the DiT.

We refer to this mechanism as *frame-wise concatenation*: it arranges clean input (conditioning) latents and noisy output latents in a single frame layout.

$$\mathbf{x}_{\text{fw}}(t) = (\{\mathbf{x}_0^m\}_{m \in \mathcal{M}_{\text{in}}}, \{\mathbf{x}_t^m\}_{m \in \mathcal{M}_{\text{out}}}). \quad (4.2)$$

$\mathbf{x}_{\text{fw}}(t)$ is the latent collection passed to the denoising model; $\mathbf{x}_0^m$ is the clean latent for modality m , $\mathbf{x}_t^m$ is the corresponding noisy output latent at diffusion timestep t , and $\mathcal{M}_{\text{in}}$ and $\mathcal{M}_{\text{out}}$ choose the input and output modalities. The equations below use frame IDs only to separate input and output roles for clarity; in principle, distinct frame-ID ranges could also mark different modalities, such as RGB and G-buffers. Following the frame-ID convention above, the noisy output latents use the first K latent-frame positions and the clean input latents use the next K positions:

$$\ell_{\text{out}} = (i, j, k), \quad \ell_{\text{in}} = (i, j, k + K), \quad 0 \leq k < K. \quad (4.3)$$

The indices ℓ_{out} and ℓ_{in} share the same spatial patch (i, j) and paired frame index k , but occupy different RoPE frame-ID positions. Inside the DiT, the latent collection is patchified into separate input and output tokens:

$$\mathbf{u}_{\ell_{\text{in}}}^{m, \text{in}} = \mathcal{P}(\mathbf{x}_{0, \ell_{\text{in}}}^m), \quad \mathbf{u}_{\ell_{\text{out}}, t}^{m, \text{out}} = \mathcal{P}(\mathbf{x}_{t, \ell_{\text{out}}}^m). \quad (4.4)$$

The map $\mathcal{P}$ patchifies each latent patch into a token. The internal frame-wise layout is the union:

$$\mathcal{U}_{\text{fw}}(t) = \left\{ \mathbf{u}_{\ell_{\text{in}}}^{m, \text{in}} : m \in \mathcal{M}_{\text{in}} \right\} \cup \left\{ \mathbf{u}_{\ell_{\text{out}}, t}^{m, \text{out}} : m \in \mathcal{M}_{\text{out}} \right\}. \quad (4.5)$$

The set $\mathcal{U}_{\text{fw}}(t)$ is the token sequence processed by DiT layers after the latents are patchified. We still write the denoising model as $f_{\theta}(\mathbf{x}_{\text{fw}}(t), t, p)$ because patchification is an internal part of the DiT. Only output latents have noise added and loss computed on them; clean input latents provide context and are discarded after the DiT pass. The corresponding masked flow-matching loss is

$$\mathcal{L}_{\text{fw}} = \mathbb{E} \left[\sum_{m \in \mathcal{M}_{\text{out}}} \|f_{\theta}(\mathbf{x}_{\text{fw}}(t), t, p)^m - (\boldsymbol{\epsilon}^m - \mathbf{x}_0^m)\|_2^2 \right]. \quad (4.6)$$

The loss $\mathcal{L}_{\text{fw}}$ sums only over $m \in \mathcal{M}_{\text{out}}$, and the superscript m on $f_{\theta}(\cdot)^m$ selects the predicted velocity for output modality m . This formulation is independent of the particular rendering direction: RGB→X uses RGB as clean input and G-buffers as noisy outputs, while X→RGB reverses the roles and may use multiple G-buffer inputs.

4.2 RGB→X models for inverse rendering

4.2.1 Preliminary RGB→X model

In generative inverse rendering (RGB→X), our goal is to take RGB frames $\mathbf{I}$ as input and produce G-buffer frames as output. The G-buffer modalities considered in our models include albedo $\mathbf{a}$, normal $\mathbf{n}$, depth $\mathbf{z}$, material properties $\mathbf{m}$ (such as roughness, metallicity, or transparency), diffuse irradiance $\mathbf{E}$, and direct lighting $\mathbf{d}$. Using the frame-wise concatenation formulation from Section 4.1.1, our preliminary RGB→X model uses RGB as input and predicts one G-buffer modality at a time:

$$\mathcal{M}_{\text{in}} = \{\mathbf{I}\}, \quad \mathcal{M}_{\text{out}} \in \{\{\mathbf{a}\}, \{\mathbf{n}\}, \{\mathbf{z}\}, \{\mathbf{m}\}, \{\mathbf{E}\}, \{\mathbf{d}\}\}. \quad (4.7)$$

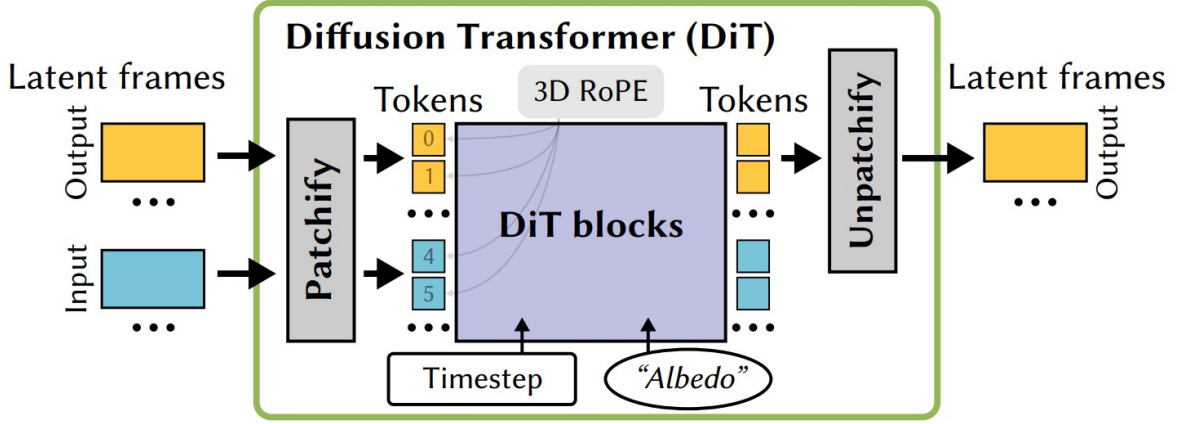


Figure 4.3: The preliminary RGB→X model concatenates the output X and input RGB, distinguished by frame ID. Input tokens have no noise added and no loss computed on them. The text prompt is repurposed for output modality selection, e.g. "albedo", "normal", etc.

The desired output modality is selected by finetuning the text prompt as a switch [18], e.g., using "albedo" or "normal" as the prompt.

This construction is intentionally minimal; see Fig. 4.3. The frame-ID convention indicates which frames are clean input frames versus noisy output frames, and the prompt indicates the desired output modality. These indirect signals are sufficient to obtain a preliminary RGB→X model, but they leave role (input/output) and modality only weakly specified. The next Section 4.2.2 replaces these implicit signals with explicit token-level mechanisms.

4.2.2 Improved RGB→X model

Frame-wise concatenation already separates clean inputs from noisy outputs, but the baseline identifies these roles only through the frame-ID convention in Eq. (4.3). A better approach than using the frame-ID to distinguish input and output tokens is to embed this information directly into tokens. For each discrete combination of token role and modality, we define a token type $\tau = (r, m)$, where role $r \in \{\text{in}, \text{out}\}$ and m is the

modality. Instead of directly adding a type embedding to the tokens, we train the model to add learned embeddings directly to the query and key vectors in the DiT layers; see Fig. 4.4 (a). For a token $\mathbf{u}_\ell^\tau$, each attention layer applies RoPE as usual and then adds learned type offsets to the attention queries and keys:

$$\mathbf{Q}_\ell^\tau = \mathcal{R}_\ell(W_Q \mathbf{u}_\ell^\tau) + \mathbf{e}_Q^\tau, \quad \mathbf{K}_\ell^\tau = \mathcal{R}_\ell(W_K \mathbf{u}_\ell^\tau) + \mathbf{e}_K^\tau. \quad (4.8)$$

The vectors $\mathbf{e}_Q^\tau$ and $\mathbf{e}_K^\tau$ are learned additive query/key type embeddings, and $\mathcal{R}_\ell$ is the RoPE transform for token index ℓ . Thus RoPE injects position through a rotation, while QK type embedding injects role and modality through additive offsets, e.g., clean RGB input versus noisy albedo output. We show that this modification results in faster finetuning convergence. We call this modification *QK type embeddings*.

Furthermore, we note that in a standard DiT [11], the same time step embedding is applied through DiT blocks to all tokens. This makes sense, as all tokens are being denoised at the same noise level. However, in our case, some tokens correspond to input frames, which are provided without noise. We therefore modify the timestep embedding to provide $t = 0$ to the tokens corresponding to input frames; see Fig. 4.4 (b). Equivalently, each token receives an effective timestep

$$\tilde{t}_\ell^\tau = \begin{cases} 0, & r(\tau) = \text{in}, \\ t, & r(\tau) = \text{out}. \end{cases} \quad (4.9)$$

The function $r(\tau)$ extracts the input/output role from the token type, and $\tilde{t}_\ell^\tau$ is the timestep embedding used for that token. An input token is therefore embedded as a clean latent, while an output token keeps the current diffusion timestep. This modification, which we call *clean input tokens*, further improves quality and finetuning convergence.

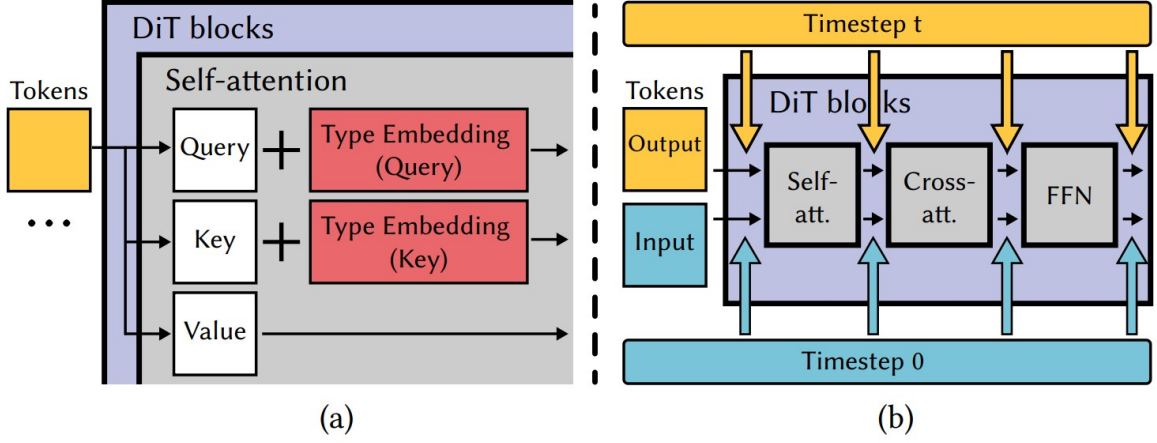


Figure 4.4: (a) Our $Q(k)$ type embedding encodes modality and input/output by adding learned offsets to the query and key vectors. (b) Our *clean token* technique modifies the timestep embedding mechanism to always feed timestep 0 (i.e., no noise) to the input tokens, resulting in faster and higher-quality finetuning.



Figure 4.5: Training $X \rightarrow \text{RGB}$ on synthetic data leads to the model quickly adopting the synthetic look; real data with annotated G-buffers is key to realistic $X \rightarrow \text{RGB}$ results.

Training $\text{RGB} \rightarrow X$ models only requires paired synthetic data: rendered RGB frames and the corresponding G-buffers. Several public datasets already provide such data, such as Hypersim [20] and InteriorVerse [19]. To improve data quality and diversity, we curate an internal collection of synthetic scenes. The collection covers both indoor and outdoor environments. We then render an internal dataset from these scenes. It contains 900 video sequences, each with roughly 100 frames of path-traced RGB and corresponding G-buffers.

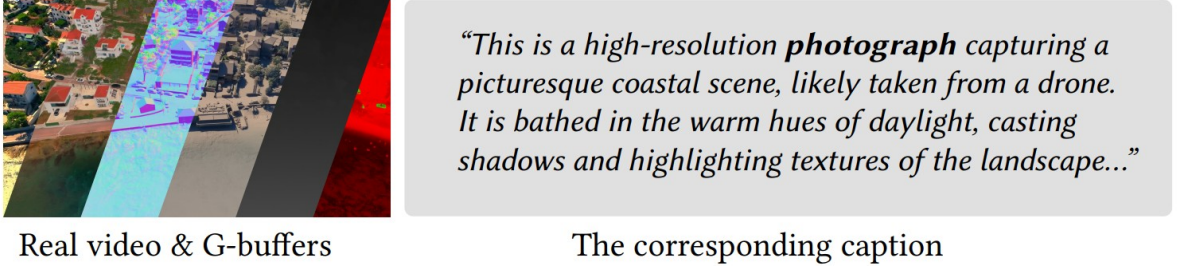


Figure 4.6: Example from our (RGB, X) dataset used to train our $X \rightarrow \text{RGB}$ models: a real video with estimated G-buffers, together with the VLM-produced caption.

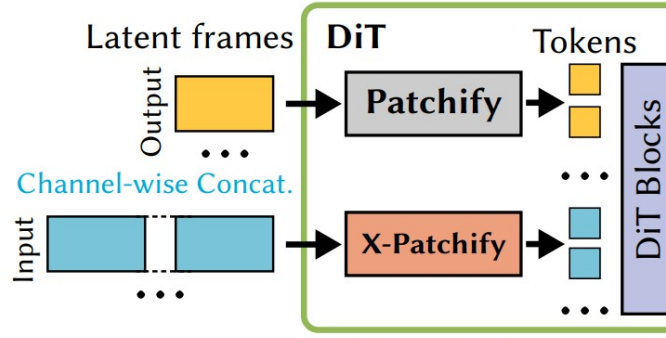


Figure 4.7: Our X -patchify technique uses channel concatenation for a flexible number of input modalities, learning a patchify operator to map all of them into the same tokens, while still keeping outputs as separate tokens.

4.3 $X \rightarrow \text{RGB}$ models for realistic generative rendering

In generative forward rendering ($X \rightarrow \text{RGB}$), our goal is to take one or more G-buffers as inputs, and produce realistic RGB frames as output, generating all information that is not included in the inputs.

To build such models, we first use our $\text{RGB} \rightarrow X$ model to prepare a dataset of real videos paired with estimated G-buffers (Section 4.3.1). We then train an $X \rightarrow \text{RGB}$ model that reuses the conditioning mechanisms introduced in previous sections, while adding a compact way to support multiple G-buffer inputs (Section 4.3.2). Finally, we describe inference-time controls for adjusting how strongly the generated RGB sequence follows the provided scene specification, including condition dropout and blurring (Section 4.3.3)

and lighting-related conditioning choices (Section 4.3.4).

4.3.1 (RGB, X) Dataset

In contrast to $\text{RGB} \rightarrow \text{X}$, it is not sufficient to train $\text{X} \rightarrow \text{RGB}$ on synthetic pairs of G-buffers and RGB images, as the results quickly shift towards a synthetic look; see Fig. 4.5. Inspired by [18], we use our $\text{RGB} \rightarrow \text{X}$ model to annotate real data with G-buffers, and train the $\text{X} \rightarrow \text{RGB}$ model on the resulting pairs. We refer to this collection of real RGB videos and corresponding estimated G-buffers as our (RGB, X) dataset. We use about 6,622 videos from the free Pexels service, holding out 371 videos for test. For each real video, we sample clips and annotate them with our $\text{RGB} \rightarrow \text{X}$ model to obtain aligned estimated G-buffers.

Furthermore, we caption the videos with Qwen2.5-VL [34]. An example is shown in Fig. 4.6. Real footage captions will generally contain "photograph, real video" etc., while rendered sequences (some of which are included in Pexels) will contain "video game, rendered" etc. in the captions. This enables style supervision while preserving the pretrained text prior, so later classifier-free guidance [8] can use the caption space to steer between realistic and synthetic looks.

4.3.2 $\text{X} \rightarrow \text{RGB}$ model with X-patchify

Following the frame-wise formulation in Section 4.1.1 and the improvements in Section 4.2.2, we provide the conditioning G-buffer frames to the model with the output RGB frames, and we use the QK type embedding and clean token techniques. In contrast to $\text{RGB} \rightarrow \text{X}$, we can now have multiple input signals (e.g. albedo and depth). We introduce the *X-patchify* module, which takes multiple input signals and packs them into the same set of input tokens; see Fig. 4.7. This is essentially using channel-wise con-

catenation for all G-buffer inputs to the model. **X**-patchify first stacks the selected clean G-buffer latents along the channel dimension, then maps each resulting patch to a single conditioning token:

$$\begin{aligned}\mathbf{x}_0^{\mathbf{X}} &= \text{concat}_{\text{ch}}(\{\hat{\mathbf{x}}_0^m\}_{m \in \mathcal{M}_{\text{in}}}), \\ \text{shape}(\mathbf{x}_0^{\mathbf{X}}) &= H' \times W' \times C_{\text{cat}} \times F', \quad C_{\text{cat}} = |\mathcal{M}_{\text{in}}|C, \\ \mathbf{u}_\ell^{\mathbf{X}, \text{in}} &= \mathcal{P}_{\mathbf{X}}(\mathbf{x}_{0,\ell}^{\mathbf{X}}).\end{aligned}\tag{4.10}$$

The operator $\text{concat}_{\text{ch}}$ denotes channel-wise concatenation, and C_{cat} is the channel count after stacking the selected C -channel G-buffer latents. The learned operator $\mathcal{P}_{\mathbf{X}}$ then patchifies this tensor into conditioning tokens. **X**-patchify is applied only to clean input signals; noisy RGB output tokens remain separate, as in Eq. (4.5). We find that this preserves quality and training convergence, unlike mixing inputs and outputs in the same set of tokens.

During inference, we use classifier-free guidance (CFG) [8] to bias **X**→**RGB** samples toward real imagery. We define a positive prompt p^+ that describes the desired realistic imagery, such as “photograph, real video, high-resolution, high-quality”, and a negative prompt p^- that describes the appearance we want to suppress, such as “synthetic, CG, rendered, video game, unrealistic”. CFG then pushes the predicted velocity from the negative-prompt prediction toward the positive-prompt prediction. We denote the **X**→**RGB** latent collection by $\mathbf{x}_{\mathbf{X} \rightarrow \mathbf{RGB}}(t)$; it contains the clean G-buffer conditions and

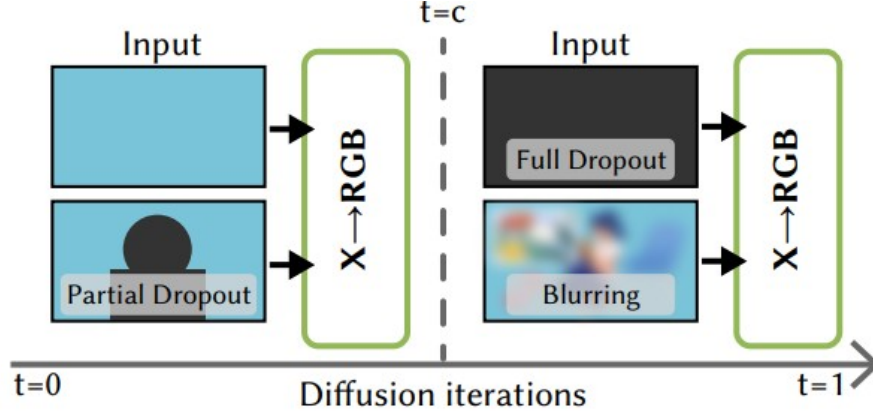


Figure 4.8: Our models can support full dropout of a G-buffer, partial dropout (where black means no guidance), and blurring. These techniques can be applied on a fraction of diffusion iterations, allowing for control in guidance strength.

the noisy RGB latent being denoised:

$$\begin{aligned}
 \mathbf{x}_{X \rightarrow \text{RGB}}(t) &= \left(\{\hat{\mathbf{x}}_0^m\}_{m \in \mathcal{M}_{\text{in}}}, \mathbf{x}_t^{\text{I}} \right), \\
 \mathbf{v}^- &= f_{\theta}(\mathbf{x}_{X \rightarrow \text{RGB}}(t), t, p^-)^{\text{I}}, \\
 \mathbf{v}^+ &= f_{\theta}(\mathbf{x}_{X \rightarrow \text{RGB}}(t), t, p^+)^{\text{I}}, \\
 \mathbf{v}_{\text{CFG}} &= \mathbf{v}^- + w (\mathbf{v}^+ - \mathbf{v}^-).
 \end{aligned} \tag{4.11}$$

The hatted latents $\hat{\mathbf{x}}_0^m$ are clean conditioning G-buffer latents, and $\mathbf{x}_t^{\text{I}}$ is the noisy RGB latent. The velocities $\mathbf{v}^+$ and $\mathbf{v}^-$ are conditioned on the positive and negative prompts, and w is the classifier-free guidance weight. The guidance direction preserves the G-buffer condition while biasing the sample away from a synthetic look and toward realism.

4.3.3 Conditions dropout and blurring

An important goal of generative rendering is to permit specifying only part of the scene, and let the model creatively fill in the rest. For this reason, we allow some input G-buffers to be blurred, specified only partially, or completely dropped during the diffusion

iterations; see Fig. 4.8. Before patchification, these choices transform each clean condition $\hat{\mathbf{x}}^m$ into the condition actually given to the model at sampling step n . Let B_σ denote Gaussian blur, $\mathbf{M}^m$ a spatial mask whose zero entries mean "no guidance" at those pixels, and $\delta_{m,n} \in \{0, 1\}$ a gate indicating whether modality m is active at step n . If n indexes the N sampling steps from coarse to fine, a condition used only for the first fraction c of sampling has

$$\delta_{m,n} = \begin{cases} 1, & n \leq cN, \\ 0, & n > cN, \end{cases} \quad \tilde{\mathbf{x}}_{(n)}^m = \delta_{m,n} \mathbf{M}^m \odot B_{\sigma_m}(\hat{\mathbf{x}}^m). \quad (4.12)$$

Here $\tilde{\mathbf{x}}_{(n)}^m$ is the transformed condition at step n , and $\odot$ denotes pointwise multiplication. The full G-buffer is removed whenever $\delta_{m,n} = 0$; spatial regions are removed where $\mathbf{M}^m = 0$; nonzero σ_m preserves the condition only at a lower spatial frequency. Setting $c = 1$ keeps the condition active throughout sampling, while smaller c applies conditioning during early coarse denoising and removes it during later detail synthesis, reducing guidance strength without changing the trained model.

To enable these controls at inference time, we expose the model to the same kinds of missing or weakened conditions during training. For training examples that use controllable G-buffer inputs, we randomly transform the clean conditions before patchification: an entire modality may be dropped, spatial regions may be masked out to indicate no guidance, or the condition may be blurred with a randomly sampled blur amount. For blurring, we sample σ from $[\sigma_{\min}, \sigma_{\max}]$, with $\sigma_{\min} = 1$ and $\sigma_{\max} = 10$. For dropout, we remove an entire modality with probability 50%, or mask out selected segment regions; in both cases, removal is implemented by setting the corresponding latent values to zero before patchification. The flow-matching loss is still computed only on the RGB output tokens; these transformations only change the clean conditioning tokens. This teaches

the model that absent, partially specified, or blurred G-buffers should be interpreted as weaker guidance rather than as literal black or blurred scene content.

4.3.4 Lighting conditions

In many cases, we would like to control the lighting instead of having it randomly sampled. One option to achieve this is by providing a *diffuse irradiance* $\mathbf{E}$ [18]. Computing diffuse irradiance requires path-tracing the scene, but this is a much simpler path-tracing problem than full rendering, as it does not require complex materials, and the Lambertian shading converges and denoises very easily.

Beyond irradiance, many other lighting representations are possible, including global IBL lighting [35] and image-space grids of local illumination probes [36]. Both of these could be added to our models given appropriate training data. We show a new alternative lighting representation, *albedo-free direct lighting* $\mathbf{d}$. Assuming a simple shading model combining one diffuse and one glossy lobe, we compute direct illumination from local as well as environment lights, and avoid multiplication by the diffuse and specular albedos when shading. This representation has the advantage of being even easier to compute than diffuse irradiance, since it does not contain multi-bounce lighting, while preserving the property of being orthogonal to the albedo buffer.

Both lighting conditions can be overly constraining when supplied at full resolution throughout the entire denoising process. We therefore use the conditioning dropout and blurring mechanisms from Section 4.3.3 to control the strength of lighting guidance: a lighting buffer can be blurred using a Gaussian with a standard deviation of σ pixels, and it can be supplied only for the first fraction $c \in [0, 1]$ of diffusion iterations.

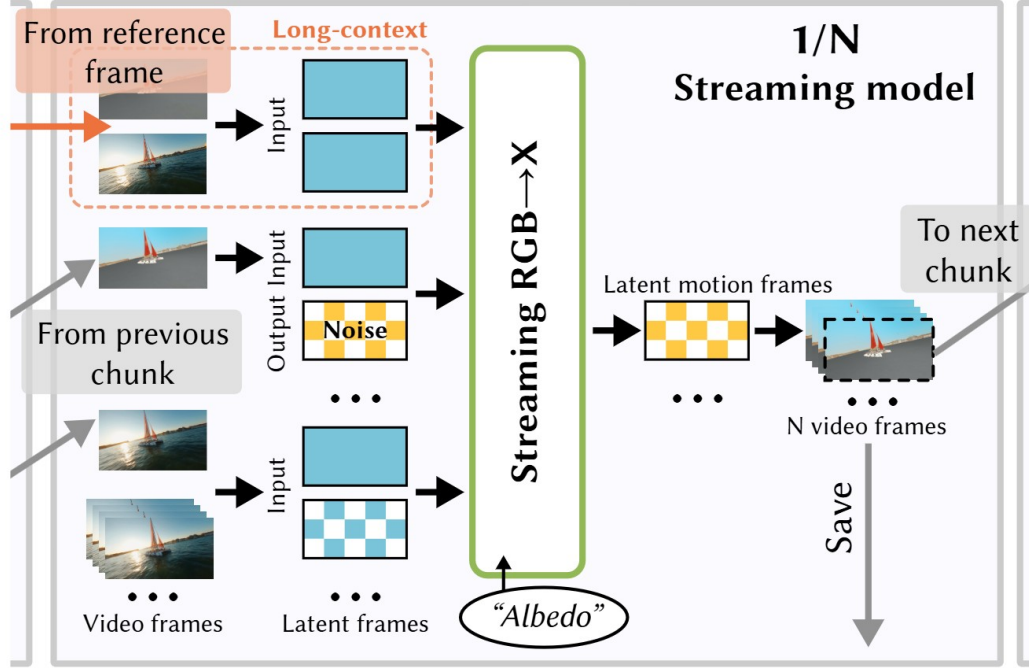


Figure 4.9: The overall design of our 1/N streaming models. For each chunk, the model receives N current input frames and one stitching frame from the previous chunk, whose input and output are already known, then produces the N new output frames through a diffusion process. Optional long-context tokens provide a clean reference frame for long-term memory. For simplicity, the figure illustrates the RGB→X case with albedo as X.

4.4 Streaming models

The models described so far operate on fixed-length video clips. This length can be increased with more training time and GPU memory, but it remains bounded by the finite context of the base text-to-video DiT model. To be truly useful for generative forward and inverse rendering, the models need to support an unbounded number of frames handled in a *streaming* manner.

Teacher forcing A straightforward idea to achieve a streaming model is to overlap one frame between chunks. We call this overlapped frame the *stitching frame*. We condition the model on the stitching frame, for which both input and output are known, and

generate outputs for N new frames, for which only inputs are provided; we term such a model a $1/N$ -streaming model. Let s index streaming chunks. Using the sampler notation from Section 4.1.1, the generated chunk is

$$\hat{\mathbf{x}}_{s,1:N}^{\text{out}} = \text{Sample}_{\theta}(\mathbf{x}_{s,1:N}^{\text{in}}, \hat{\mathbf{x}}_{s-1,N}^{\text{out}}, p). \quad (4.13)$$

The hat marks generated outputs, $\mathbf{x}_{s,1:N}^{\text{in}}$ is the clean input sequence for chunk s , $\hat{\mathbf{x}}_{s-1,N}^{\text{out}}$ is the generated stitching frame from the previous chunk, and p is the text prompt used by the diffusion sampler.

This simple approach can work for short sequences (100–200 frames), but accumulates error over longer sequences. The practice of training on ground truth while relying on past predictions at inference time is known as *teacher forcing*. As noted by [16], this leads to drift and error accumulation. With $\mathbf{x}^{\text{out},*}$ denoting ground truth, the stitching frame differs between training and inference:

$$\mathbf{x}_{\text{train}}^{\text{prev}} = \mathbf{x}_{s-1,N}^{\text{out},*}, \quad \mathbf{x}_{\text{infer}}^{\text{prev}} = \hat{\mathbf{x}}_{s-1,N}^{\text{out}}. \quad (4.14)$$

The star in $\mathbf{x}^{\text{out},*}$ denotes ground truth. Prediction errors can therefore be fed back as the next chunk’s stitching frame.

Augmentation To break the “perfection” of teacher forcing, we perturb that one stitching frame with: (1) per-channel brightness shifts up to $\pm 2\%$, (2) Gaussian blur or unsharp masking (σ up to 1.5), and (3) a VAE round-trip simulating the lossy decode-encode cycle required at inference to extract the stitching frame from decoded outputs. We write the augmented stitching frame as

$$\tilde{\mathbf{x}}^{\text{prev}} = \mathcal{A}_{\lambda}(\mathbf{x}_{s-1,N}^{\text{out},*}), \quad \lambda \in [0, 1]. \quad (4.15)$$

The operator $\mathcal{A}_\lambda$ applies the brightness, blur or sharpening, and VAE round-trip perturbations, while λ controls the augmentation strength. While this slightly reduces drifting, it does not resolve it.

Long context tokens. Another important issue in streaming designs is lack of long-term memory: even if results are locally temporally stable and do not accumulate error, conditioning only on the stitching frame can result in forgetting details of earlier frames. This can produce issues like temporally shifted albedo in $\text{RGB} \rightarrow \text{X}$ models, or slowly changing lighting in $\text{X} \rightarrow \text{RGB}$ models. Our solution, inspired by FramePack [17], is to provide *long-context tokens* to the model. For simplicity, we provide a single reference frame, though multiple reference frames could be supported by the same approach. During training, this is the first frame of the dataset sequence, though during inference this is not required. This reference frame should be representative of overall colors, lighting, etc. Long context appends reference-frame tokens to the per-chunk tokens:

$$\mathcal{U}_s^{\text{long}}(t) = \mathcal{U}_s^{\text{chunk}}(t) \cup \left\{ \mathbf{u}_\ell^{m, \text{ref}} : m \in \mathcal{M}_{\text{in}} \cup \mathcal{M}_{\text{out}} \right\}. \quad (4.16)$$

The set $\mathcal{U}_s^{\text{chunk}}(t)$ contains the regular per-chunk tokens, while $\mathbf{u}_\ell^{m, \text{ref}}$ denotes a clean reference-frame token. The reference tokens provide stable sequence-level color, lighting, and material appearance, even when the current chunk is far from the beginning of the video. This long-context feature reduces drifting and error accumulation.

Self forcing Inspired by [16], we propose to train the model on its own predictions. Specifically, we train the model to predict the flow velocity from its own partially denoised latent as well as the generated stitching frame from the previous chunk. To achieve that, we run inference from pure noise during training time, and add noise to this prediction instead of to the ground truth (and keep the prediction around for the next chunk’s

training step). Thus the noisy training latent is constructed from the model’s sampled output rather than from ground truth:

$$\begin{aligned}\hat{\mathbf{x}}_{s,0}^{\text{out}} &= \text{Sample}_{\theta}(\mathbf{x}_{s,1:N}^{\text{in}}, \hat{\mathbf{x}}_{s-1,N}^{\text{out}}, p), \\ \hat{\mathbf{x}}_{s,t}^{\text{out}} &= (1 - t)\hat{\mathbf{x}}_{s,0}^{\text{out}} + t\epsilon.\end{aligned}\tag{4.17}$$

The latent $\hat{\mathbf{x}}_{s,0}^{\text{out}}$ is a sampled output for chunk s , and $\hat{\mathbf{x}}_{s,t}^{\text{out}}$ is its noisy version used for training. The model is then trained on latents closer to those it will produce and reuse during streaming inference.

We find that self forcing by itself is not immediately better than teacher forcing: it can struggle to converge and may replace drift with flickering between chunks. A hybrid strategy works better: first train a teacher forcing model with the above augmentations, then continue from that checkpoint with an interleaved teacher forcing and self forcing stage. In this second stage, each training example uses teacher forcing or self forcing with equal probability, and we use a 10× smaller learning rate to stabilize training. This strategy helps because the model needs to be already reasonably good to produce valid samples during self forcing training. Note that unlike [16], we do not currently use DMD distillation [37, 38] or adversarial losses. Another difference is that their model is an $L/1$ model, where L is up to 81; that is, conditioning on all previous frames in an 81-frame generation.

4.5 Unified model

The techniques introduced in previous sections (frame-wise concatenation, QK type embedding, and clean tokens) generalize to a single unified RGB×X model, that is, a single model capable of generative rendering and inverse rendering. We simply concatenate all input sequences and all desired output sequences as frames, mark which is which

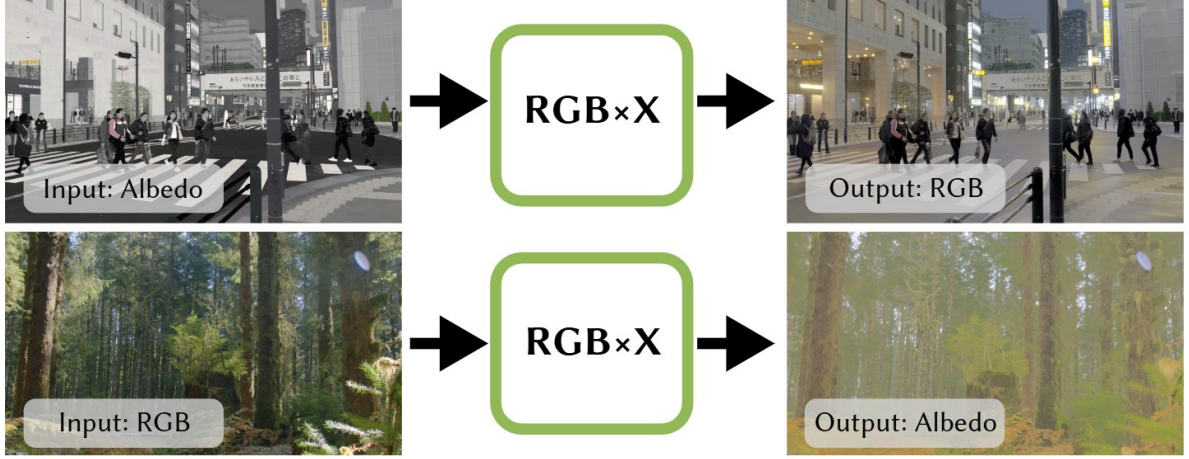


Figure 4.10: Our proof-of-concept unified model, $\text{RGB} \times \text{X}$, supports forward and inverse rendering within the same model weights. The input and output tokens are marked using embeddings, just like modalities, letting the model infer the desired task.

through embeddings, and train the model with the desired combinations. For each training example, choosing $\mathcal{M}_{\text{in}}$ and $\mathcal{M}_{\text{out}}$ specifies both the latent collection and the internal token layout:

$$\begin{aligned} \mathbf{x}_{\text{uni}}(t; \mathcal{M}_{\text{in}}, \mathcal{M}_{\text{out}}) &= (\{\mathbf{x}_0^m\}_{m \in \mathcal{M}_{\text{in}}}, \{\mathbf{x}_t^m\}_{m \in \mathcal{M}_{\text{out}}}), \\ \mathcal{U}_{\text{uni}}(t; \mathcal{M}_{\text{in}}, \mathcal{M}_{\text{out}}) &= \{\mathbf{u}_{\ell}^{m, \text{in}} : m \in \mathcal{M}_{\text{in}}\} \cup \{\mathbf{u}_{\ell, t}^{m, \text{out}} : m \in \mathcal{M}_{\text{out}}\}. \end{aligned} \quad (4.18)$$

The collection $\mathbf{x}_{\text{uni}}$ is the external latent input to the model, while $\mathcal{U}_{\text{uni}}$ is the internal token layout after patchification. A task is therefore a choice of which modalities are clean conditions and which modalities are noisy outputs to denoise. As in the frame-wise model, training ignores input tokens and evaluates flow matching only on output modalities:

$$\mathcal{L}_{\text{uni}} = \mathbb{E} \left[\sum_{m \in \mathcal{M}_{\text{out}}} \|f_{\theta}(\mathbf{x}_{\text{uni}}(t), t, p)^m - (\epsilon^m - \mathbf{x}_0^m)\|_2^2 \right]. \quad (4.19)$$

The loss $\mathcal{L}_{\text{uni}}$ is the output-only flow-matching loss for the unified model, and $\mathbf{x}_{\text{uni}}(t)$ denotes the latent collection for the sampled task. The type embeddings identify role and

modality, allowing the same weights to express $\text{RGB} \rightarrow \text{X}$, $\text{X} \rightarrow \text{RGB}$, or multi-input/multi-output configurations.

One difference is that we can no longer use the X -patchify module introduced in Section 4.3.2 to patchify multiple output signals into one single set of tokens; when these tokens are noisy tokens, the model struggles to converge. Therefore, we need a separate set of tokens per modality. As shown in Fig. 4.10, our unified $\text{RGB} \times \text{X}$ model can achieve both forward and inverse rendering, and support multiple inputs and outputs. In summary, techniques introduced in previous sections also generalize to a unified model that supports inverse and forward generative rendering, though we leave a full exploration of model unification for future work.

4.6 Results

4.6.1 Implementation details

The base model used in our experiments is the Wan 2.1 14B video model [12]. Unless stated otherwise, we finetune all DiT parameters of Wan 2.1 while keeping the VAE frozen. We use AdamW with a constant learning-rate schedule, learning rate 10^{-5} for image/video finetuning and 10^{-6} for streaming finetuning. Training uses a global batch size of 128 across 32 A100 80GB GPUs. All reported inference timings are measured on a single A100 80GB GPU.

RGB $\rightarrow$ X image and video models. We train our RGB $\rightarrow$ X models on the internal synthetic dataset described in Section 4.2.2. We use a two-step training strategy. We first finetune the base model on still images rendered at 1920×1088 for 200 steps; each image step takes 70 seconds. We then finetune the resulting image model on 17-frame videos at 1280×720 for 500 steps; each video step takes 200 seconds.

X→RGB image and video models. We train several X→RGB models on the (RGB, X) dataset described in Section 4.3.1, each with a different set of conditioning G-buffers. The specialized models shown in Fig. 4.13 each use one of the following conditioning sets: albedo, normal, depth, albedo with irradiance, or albedo with direct lighting. We also train a generalized X→RGB model conditioned on albedo, normal, irradiance, and material inputs with modality dropout for comparison with RGB↔X [18]. For all X→RGB training, we downsample real videos to 1280×720 , sample 17-frame clips, and extract G-buffers using our RGB→X model. We use a two-step strategy: first finetuning on still frames for 100 steps, then finetuning on 17-frame videos for 200 steps. The generalized model instead uses 100 image steps followed by 700 video steps to improve convergence.

Streaming models. In our experiments, we instantiate the $1/N$ streaming formulation as a $1/16$ streaming model. Both RGB→X and X→RGB streaming models are finetuned from their corresponding 17-frame video models. Streaming finetuning has two stages. In the first stage, we train with augmented teacher forcing; each step has a similar cost to the corresponding video-model training step. The augmentation intensity in Eq. (4.15) ramps linearly over 100 training steps, with 20% of samples left unaugmented. We train this first stage for 80 steps for RGB→X and 40 steps for X→RGB. In the second stage, we train on 113-frame sequences with hybrid teacher forcing and self forcing examples; each training step operates on a 17-frame chunk and takes 1000 seconds. We interleave teacher forcing and self forcing with equal probability, and train this hybrid stage for another 80 steps for RGB→X and 40 steps for X→RGB.

Inference. All results are generated using 20 diffusion sampling steps. For 17-frame generation, RGB→X takes approximately 200 seconds to synthesize a 17-frame video segment with CFG = 1, whereas X→RGB takes approximately 400 seconds with CFG =

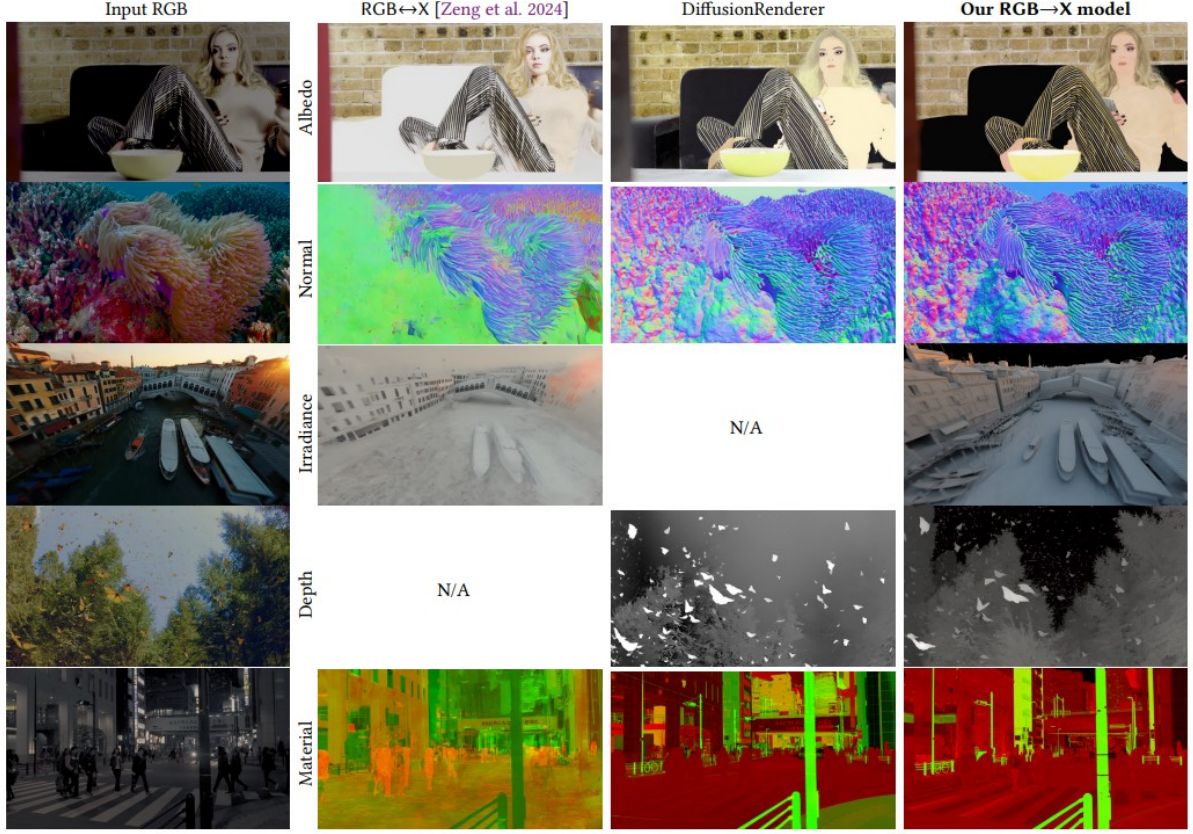


Figure 4.11: Qualitative RGB→X comparison on (RGB, X) test set. Columns show the input RGB, RGB↔X [18], DiffusionRenderer [35] (specifically its newer version based on Cosmos [14]), and ours. Our model produces flatter albedo with less residual shading, supports both depth and irradiance, and avoids the sky-depth failure shown by DiffusionRenderer; DiffusionRenderer produces normals of similar visual quality.

3. For the streaming variant, the total inference time is approximately the above inference time multiplied by the number of chunks.

4.6.2 RGB→X results

We first evaluate our RGB→X model. The supplementary video shows temporally coherent albedo estimates (0:10–1:09) and irradiance estimates (1:12–2:12), both central targets of intrinsic decomposition. Normals, depth, and material estimates are shown in 2:18–2:44.

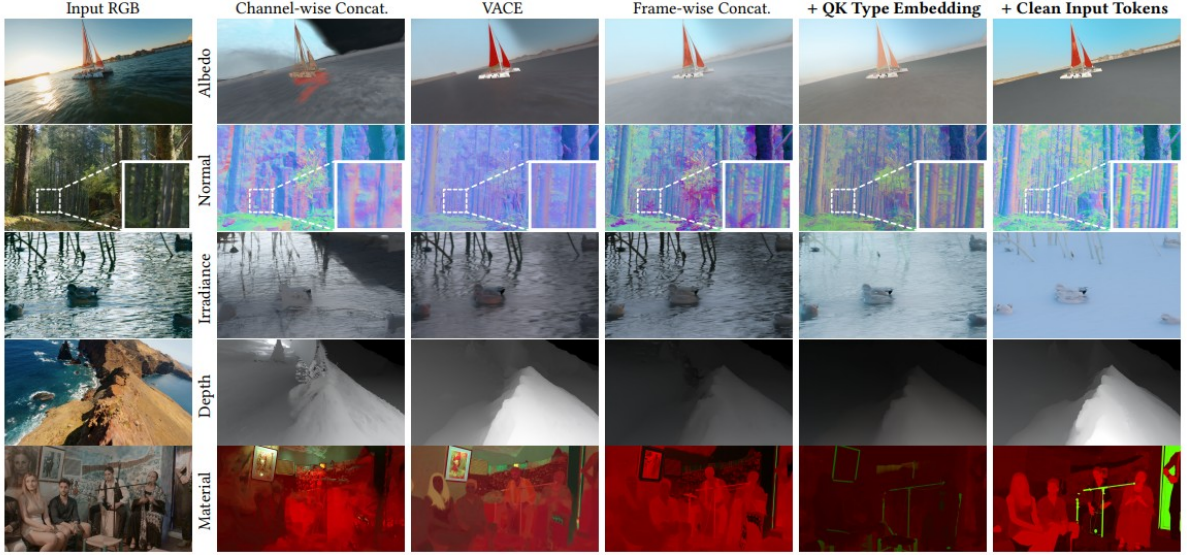


Figure 4.12: Ablation of RGB→X conditioning choices and our proposed techniques. We compare channel-wise concatenation and VACE [33] with the preliminary frame-wise model, then add QK type embeddings and clean input tokens. All variants are trained for the same duration.

Comparisons In Fig. 4.11, we visually compare our G-buffer estimates to RGB↔X [18] and the Cosmos-based version of DiffusionRenderer [35, 14], using the official checkpoints released by the authors. On held-out (RGB, X) test set, RGB↔X produces less accurate estimates, consistent with its older Stable Diffusion 2.1 backbone and indoor-only training data. DiffusionRenderer produces normals of similar visual quality, but its albedo estimates retain more residual shading and its depth prediction fails on sky regions. The methods also differ in modality coverage: RGB↔X does not support depth, while DiffusionRenderer does not support irradiance. Our model therefore enables irradiance estimation on real video footage in this comparison; see the supplementary video (1:11–2:12), with additional visual comparisons at 2:51–3:32. In Fig. 4.12, we also compare conditioning mechanisms against channel-wise concatenation and VACE [33]. Channel-wise concatenation, used in RGB↔X and DiffusionRenderer, extends patchification to two concatenated latent frames: a clean input and a noisy output. This uses fewer tokens, but convergence is significantly worse because the model must quickly repurpose

PSNR	Albedo	Normal	Irradiance	Depth
RGB $\leftrightarrow$ X	18.39	19.25	11.66	N/A
DiffusionRenderer	15.29	21.54	N/A	17.72
Ours	20.17	21.22	25.19	29.47
SSIM	Albedo	Normal	Irradiance	Depth
RGB $\leftrightarrow$ X	0.7252	0.6901	0.7162	N/A
DiffusionRenderer	0.7425	0.7372	N/A	0.6570
Ours	0.8219	0.7638	0.8589	0.9032
LPIPS	Albedo	Normal	Irradiance	Depth
RGB $\leftrightarrow$ X	0.1920	0.2225	0.2673	N/A
DiffusionRenderer	0.2675	0.2275	N/A	0.3309
Ours	0.1420	0.1986	0.3535	0.1166

Table 4.2: Quantitative comparison of our RGB $\rightarrow$ X results to RGB $\leftrightarrow$ X [18] and DiffusionRenderer [35] on the test set of the Hypersim dataset. Our result is best across all three metrics for depth and albedo, and in two out of three metrics on normal and irradiance.

attention patterns learned between video frames into attention between input and output signals. VACE instead injects input signals through control tokens.

Quantitative comparisons In Table 4.2, we quantitatively compare our RGB $\rightarrow$ X results to previous methods on the Hypersim test set [20]. Our model is best across all three metrics for albedo and depth, and best in two out of three metrics for normal and irradiance. Since no publicly available dataset provides reliable ground-truth roughness and metallicity buffers, we do not evaluate material properties quantitatively. Note that RGB $\leftrightarrow$ X [18] was trained on Hypersim whereas our model was not. Hypersim is also synthetic, so improvements on real inputs are better judged visually.

Ablations In Fig. 4.12, we also ablate the two key techniques introduced in Section 4.2.2. Starting from the preliminary frame-wise RGB $\rightarrow$ X model described in Section 4.2.1, adding QK type embeddings improves the estimates across modalities, and adding clean input tokens further improves quality and convergence.

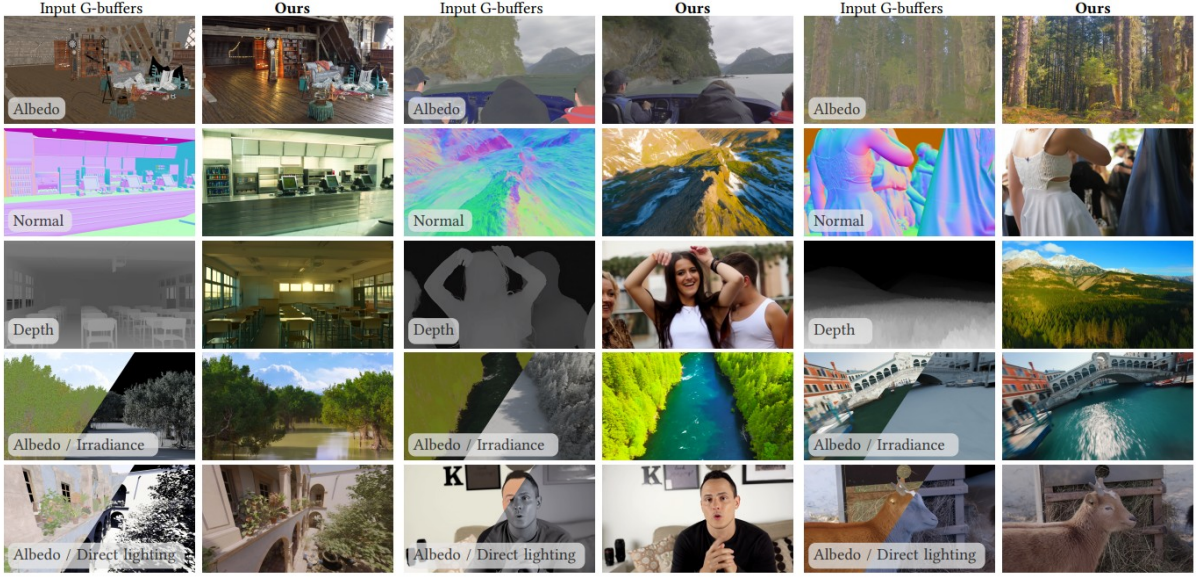


Figure 4.13: Results from various $X \rightarrow \text{RGB}$ models, conditioned on albedo, normal, depth, and combinations of albedo with irradiance (blurred and dropped after the first 20% diffusion iterations), and albedo with albedo-free direct lighting. The first column uses synthetic G-buffers rendered from 3D scenes; the remaining columns use G-buffers estimated from real videos. All models support video; please see supplementary video for many more video results.

4.6.3 $X \rightarrow \text{RGB}$ results

In Fig. 4.13, we illustrate specialized $X \rightarrow \text{RGB}$ models trained to condition on different G-buffer inputs. The first three rows use separate models trained to condition on albedo, normal, and depth. The last two rows show lighting-guided generation from models trained to condition on albedo with irradiance, where irradiance is blurred and dropped after the first 20% of denoising iterations, and albedo with direct lighting. The first column uses synthetic G-buffers rendered from 3D scenes, while the remaining examples use G-buffers estimated from real videos by our $\text{RGB} \rightarrow X$ model. Static images only partially capture the model capabilities, so we refer readers to the supplementary video for additional results: 3:36–7:11 for albedo guidance, including multiple seeds and text-prompt comparisons; 7:16–7:33 for depth guidance; 7:43–8:11 for normal guidance; 8:18–8:40 for albedo with irradiance; and 8:49–9:20 for albedo with direct illumination.

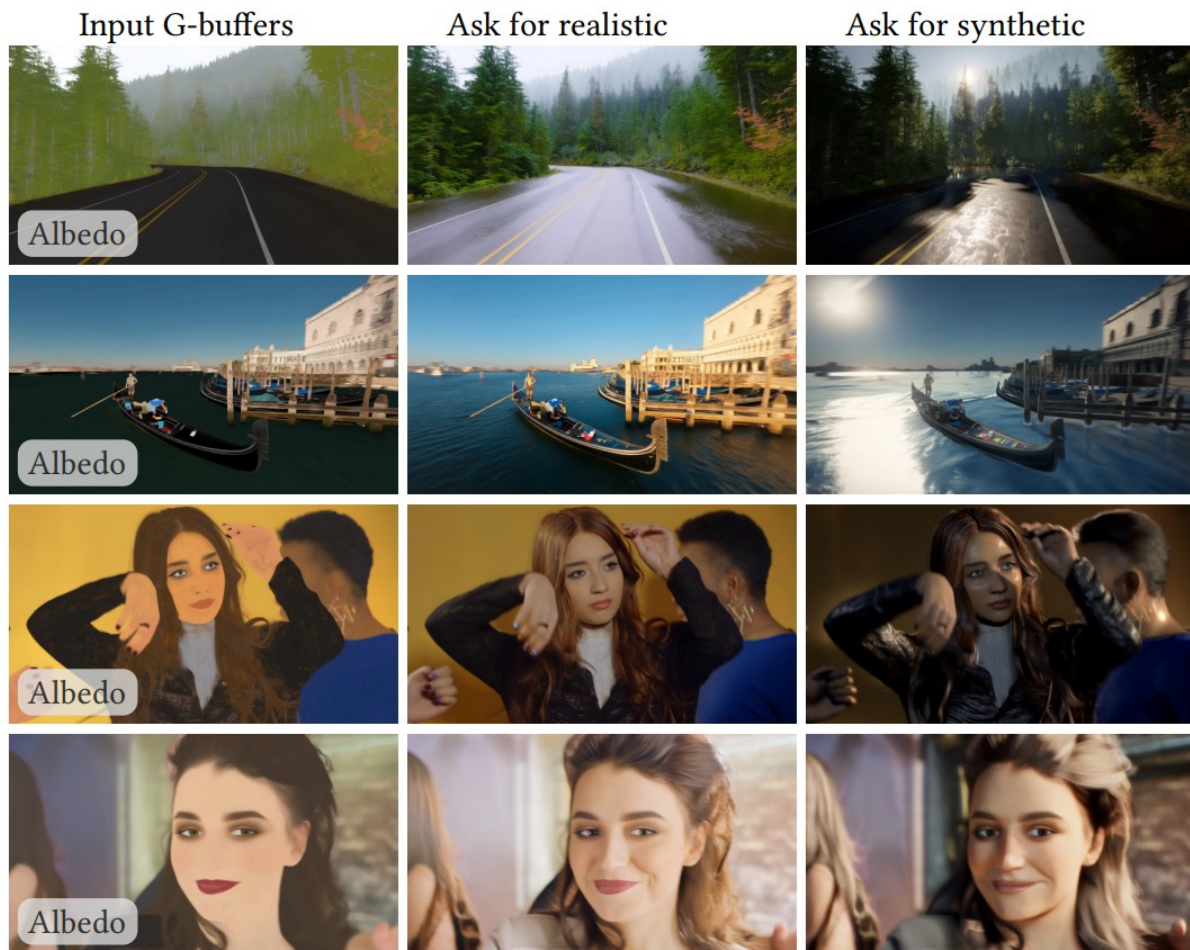


Figure 4.14: Realism control with text guidance. We show input albedo buffers (left), CFG with a positive prompt for realistic imagery and a negative prompt for synthetic imagery, pushing towards realism (middle), and the same prompts flipped, producing a more synthetic appearance (right).

Realism control In Fig. 4.14, we show how the positive and negative prompts described in Section 4.3.2 control the realism of $X \rightarrow \text{RGB}$ outputs. With a positive prompt asking for realistic imagery and a negative prompt describing synthetic or rendered appearance, CFG pushes the samples toward photographic lighting, textures, and camera-like image statistics. Flipping the two prompts produces the opposite effect: the geometry and colors still follow the input albedo buffers, but the generated images become more synthetic and stylized.

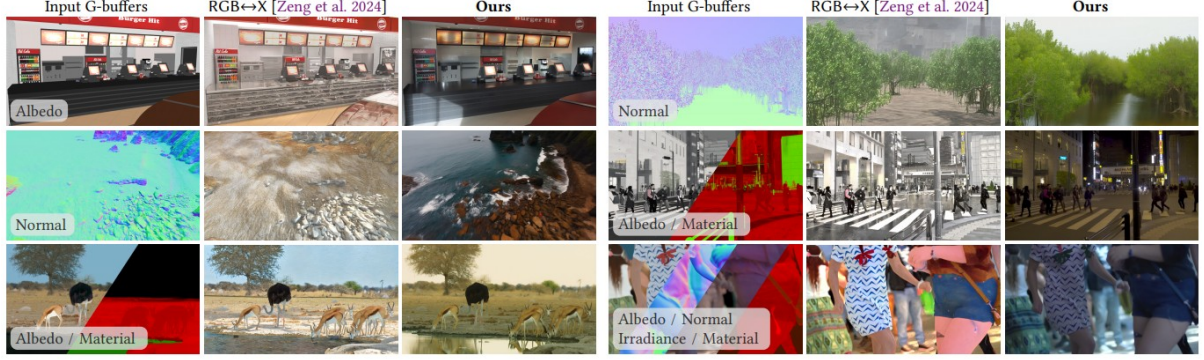


Figure 4.15: Qualitative $X \rightarrow \text{RGB}$ comparison with $\text{RGB} \leftrightarrow X$ [18]. The first row uses synthetic G-buffers rendered from 3D scenes; the remaining rows use G-buffers estimated from real videos. Compared with $\text{RGB} \leftrightarrow X$, our results appear less synthetic. Since $\text{RGB} \leftrightarrow X$ [18] is an image model, its outputs also exhibit temporal inconsistencies; please refer to the supplementary video.

Comparisons In Fig. 4.15, we compare our generalized $X \rightarrow \text{RGB}$ model, trained to condition on albedo, normal, irradiance, and material inputs with modality dropout, with [18]. Results in the first row use synthetic G-buffers rendered from 3D scenes, while the other rows use estimated G-buffers from real videos. Outputs from $\text{RGB} \leftrightarrow X$ [18] often appear unrealistic and flat compared to ours. We attribute this improvement to the stronger DiT video backbone and to training on realistic videos annotated with estimated G-buffers, rather than mixing synthetic and real RGB supervision. Further comparisons are shown in the supplementary video; since $\text{RGB} \leftrightarrow X$ [18] is an image model, its outputs also exhibit temporal inconsistencies. In Fig. 4.16, we further compare $X \rightarrow \text{RGB}$ with albedo conditioning against $\text{RGB} \leftrightarrow X$ [18], VACE [33], and a channel-wise concatenation baseline. A direct $X \rightarrow \text{RGB}$ comparison with DiffusionRenderer [35] is not straightforward because DiffusionRenderer requires all G-buffers to be fed to the model, including a static image-based lighting (IBL) input, and it is unclear whether the IBL is defined around the camera or over the entire scene. Since DiffusionRenderer uses channel-wise concatenation for conditioning, we instead include a channel-wise concatenation baseline trained on our data, in addition to VACE and $\text{RGB} \leftrightarrow X$ [18]. All methods except $\text{RGB} \leftrightarrow X$ [18] are trained

Method	FID ↓
Channel-wise concat.	88.6039
VACE [33]	62.2883
RGB $\leftrightarrow$ X [18]	62.3884
Ours	45.3871

Table 4.3: FID comparison for $X \rightarrow \text{RGB}$ on the held-out (RGB, X) test set. We compute FID between generated frames and real video frames from the test set. Lower is better.

on our (RGB, X) dataset from the same base model for the same duration. Our model produces the most realistic results while following the input albedo. VACE sometimes fails to follow the input albedo and produces less realistic results. RGB $\leftrightarrow$ X [18] outputs are unrealistic and flat. Channel-wise concatenation can produce realistic results from the base model’s prior, but it often fails to follow the albedo.

Quantitative comparisons In Table 4.3, we quantitatively compare $X \rightarrow \text{RGB}$ with albedo conditioning on the held-out (RGB, X) test set. We run each model on the same test split, which contains 6307 real video frames, and compute Fréchet Inception Distance (FID) [39] against the real-frame distribution of the test set. Our model achieves the lowest FID, improving substantially over VACE, RGB $\leftrightarrow$ X, and the channel-wise concatenation baseline, which is consistent with the qualitative realism differences shown in Fig. 4.16.

Additional prompt control In Fig. 4.17, we show that text prompts can further control lighting and material appearance even when the G-buffer input is fixed. With the default prompt, our albedo-conditioned model follows the albedo tone but misses the warm ambient lighting from the yellow strip lights, because the albedo buffer does not encode emissive surfaces. Adding a scene-specific prompt, “yellow strip lights create warm ambient light, metal dancing robot at center”, produces warmer illumination and a glossier robot, closely matching the path-traced reference.

Spatial dropout In Fig. 4.18, we illustrate spatial conditioning dropout, which allows the user to specify albedo only in selected regions and let the model generate content freely elsewhere. The left column shows sparse input G-buffers with zero-valued regions that mean “no guidance” rather than “black albedo”. The baseline $X \rightarrow \text{RGB}$ model, which is not trained with segment dropout, tends to produce flat-looking visuals in these unguided regions (middle column). Training with randomly dropped segments allows the user to provide spatially sparse signals without losing visual richness in the unconditioned regions (right column). To train this model, we run the SAM 2 video segmentation model [40] on the (RGB, X) training sequences, and randomly zero out albedo in some segments during training.

4.6.4 Streaming results

In Fig. 4.19, we illustrate our $\text{RGB} \rightarrow X$ streaming results, as well as ablations comparing to simpler variants. In (a) we illustrate the drifting issues of teacher forcing (training using ground truth data and inference using predicted data from the previous chunk). In (b), we apply augmentations of color and brightness in an attempt to break the perfection of baseline teacher forcing; this leads to only a slight improvement. In (c), we illustrate the more significant improvement from conditioning on a long-context frame (typically but not necessarily the first frame of the sequence). However, serious drifting still eventually becomes apparent on 600+ frame sequences. Finally, in (d) we show our final streaming $\text{RGB} \rightarrow X$ model, combining the previous improvements with a mixed self forcing / teacher forcing training strategy. This approach leads to stable estimation without catastrophic drifting for about 1000 frames.

In Fig. 4.20, we show the results of our streaming $X \rightarrow \text{RGB}$ model trained to condition on albedo sequences, applied to synthetic (top) and estimated (bottom) albedo inputs.

Despite not being conditioned on lighting, this model shows good temporal coherence and lighting consistency. Please see the supplementary zip file with an HTML page containing more streaming RGB $\rightarrow$ X and X $\rightarrow$ RGB videos.

4.6.5 Discussion and limitations

The optimality of G-buffers as a form of generative renderer guidance remains to be confirmed by further research, but we believe our experiments provide promising initial evidence, such as our extensive X $\rightarrow$ RGB results with albedo and lighting conditioning shown in the video. This form of guidance is capable of controlling geometry, color and lighting, while leaving sufficient space for generative model creativity.

One limitation is that our models are currently no faster than the original, large Wan 2.1 14B model they are finetuned from. Our models could be seen as *teacher models*, and could be distilled to fast interactive models by adapting DMD or similar teacher-student techniques. Rendering G-buffers is certainly fast, but full global illumination may not be; future GPUs are likely to have more neural inference power and a distilled model based on our framework may soon become faster than classical Monte Carlo rendering, in addition to the benefits of learned realism priors.

Our single-frame long-context solution could be extended to a whole video sequence or a true 3D context to provide improved memory. Our per-segment spatial dropout could be extended further, where some segments could be partially controlled, e.g., through a neural descriptor of the desired object / material, rather than having to specify either per-pixel detail or nothing. Finally, our models can often achieve realistic X $\rightarrow$ RGB results even from fairly coarse and flat G-buffer content, but the right approach to produce 3D content for best results with generative rendering is a completely unstudied problem.

4.7 Summary

We introduced RGBX-Next, a unified framework for generative forward and inverse rendering, capable of estimating G-buffers from images, videos and streams, and rendering realistic RGB outputs from G-buffer inputs. Our key contribution is a general recipe for finetuning diffusion transformer (DiT) models into generative forward and inverse renderers.

We use frame-wise concatenation to separate clean conditioning tokens from noisy output tokens, and add QK type embeddings and clean input tokens to make each token’s role, modality, and noise level explicit. For $X \rightarrow \text{RGB}$ models, we use X-patchify to pack multiple G-buffer inputs into one compact conditioning stream. We further use prompt guidance, condition blurring and dropout, spatial dropout, and lighting buffers to provide flexible control over realism, appearance, and illumination.

Our experiments show strong results in both directions: $\text{RGB} \rightarrow X$ models recover temporally coherent G-buffers from RGB videos, while $X \rightarrow \text{RGB}$ models synthesize realistic RGB videos from synthetic or estimated G-buffer inputs. We further extend both directions to streaming by combining teacher forcing, self forcing and long-context techniques. We believe our trained models, as well as the overall framework introduced in this paper, can be useful for future progress in generative forward and inverse rendering.

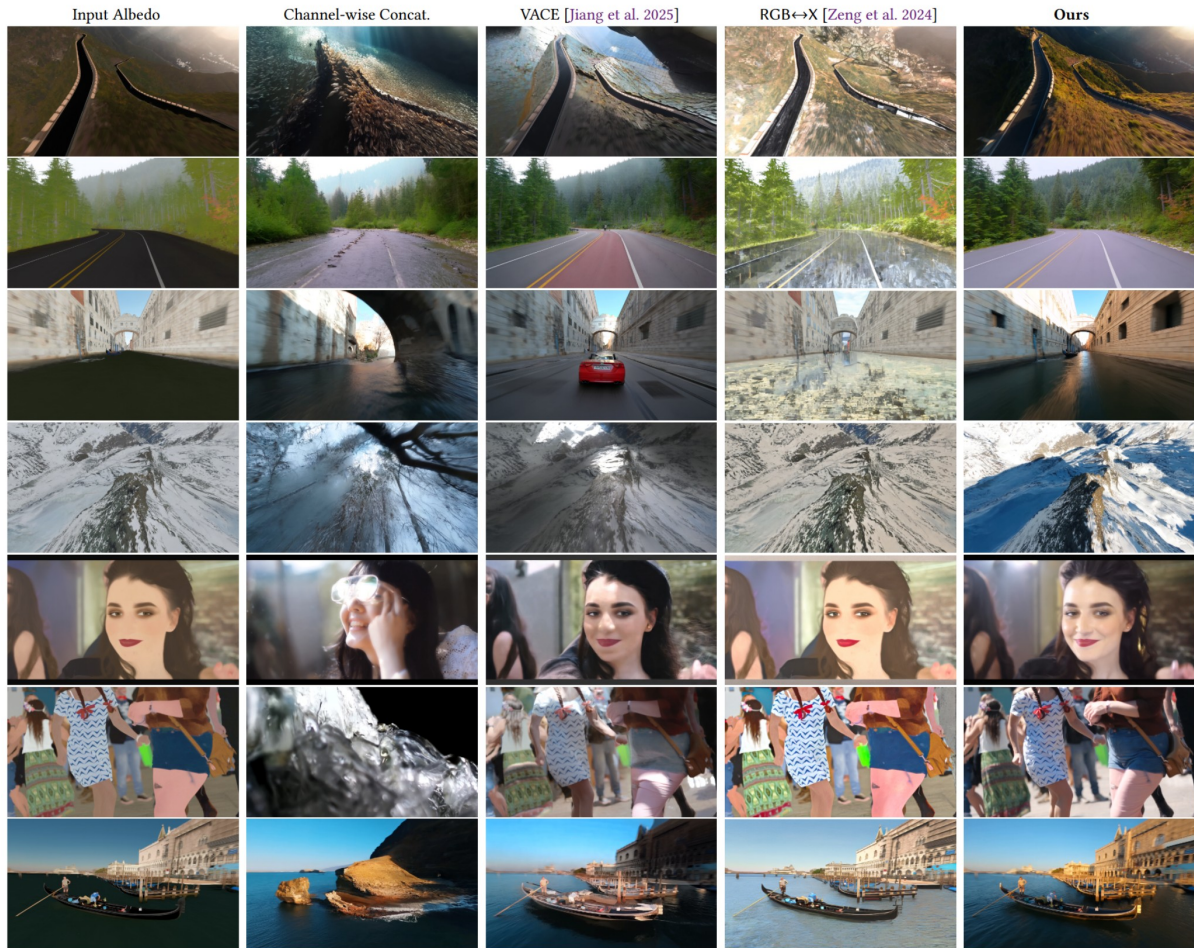


Figure 4.16: Qualitative $X \rightarrow \text{RGB}$ comparison with albedo conditioning against $\text{RGB} \leftrightarrow X$ [18], VACE [33], and a channel-wise concatenation baseline, which follows the conditioning mechanism used by DiffusionRenderer [35]. All methods except $\text{RGB} \leftrightarrow X$ are trained on our (RGB, X) dataset from the same base model for the same training time. VACE and channel-wise models produce broadly realistic results but do not follow the guidance closely, while $\text{RGB} \leftrightarrow X$ shows less realism due to being trained on exclusively indoor scene content.

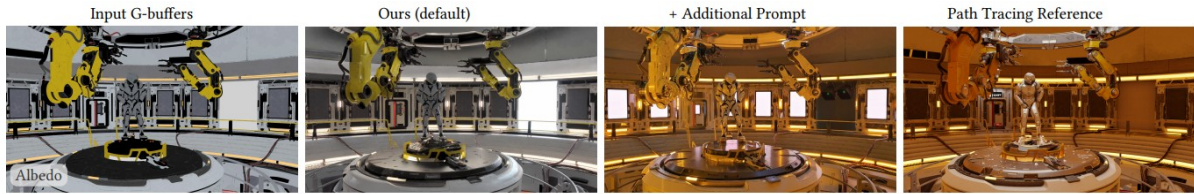


Figure 4.17: Additional text prompts can control lighting and material appearance without additional G-buffer inputs. With the default prompt (“Photograph, high-resolution, 4K, high-quality”), the result follows the albedo tone but misses the warm ambient light from the yellow strip lights. Appending “yellow strip lights create warm ambient light, metal dancing robot at center” better matches the path-traced reference and shifts the robot from diffuse to glossy.



Figure 4.18: Spatial conditioning dropout for sparse albedo guidance. We show sparse input albedo, where zero-valued regions mean “no guidance” rather than black albedo (left), the baseline $X \rightarrow \text{RGB}$ model without segment-dropout training, which produces flat visuals in these regions (middle), and our segment-dropout model, which preserves visual richness in unguided regions (right).

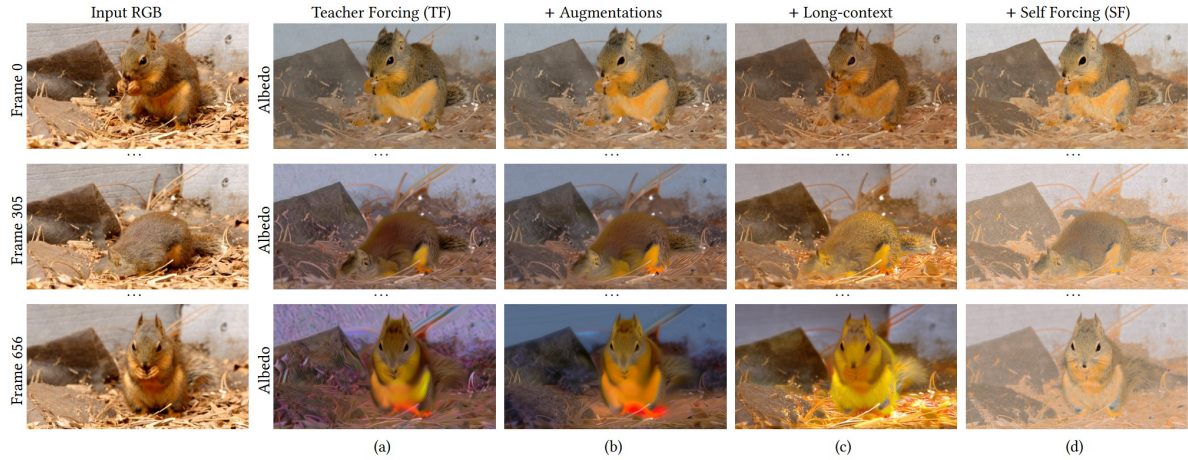


Figure 4.19: Ablation for 1/16 streaming RGB→X. Teacher forcing trains on ground-truth stitching frames but uses previous predictions at inference, leading to blur and error accumulation after roughly 100–200 frames (a). Stitching-frame augmentations help only slightly (b), and long-context tokens improve color preservation but do not eliminate drift (c). Combining these components with hybrid self forcing training yields stable estimates over hundreds of frames (d).

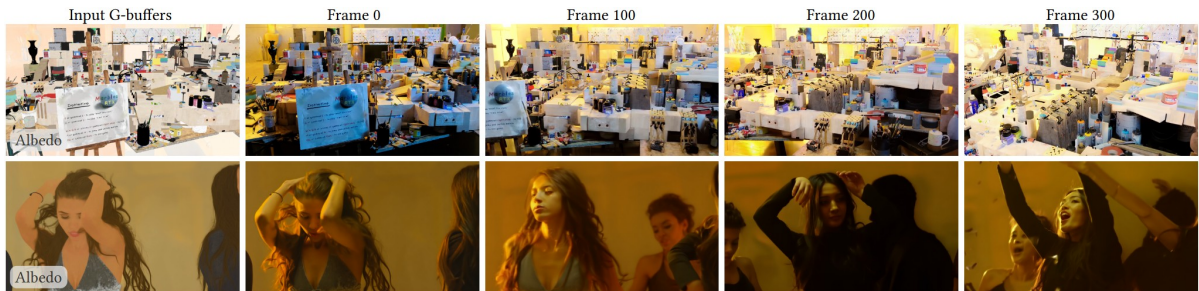


Figure 4.20: Streaming X→RGB results with self forcing training, applied to synthetic albedo sequences (top) and estimated ones from real videos (bottom). The generated videos maintain temporal coherence and lighting consistency across frames; full sequences are provided in the supplementary material.

Chapter 5

Conclusion and Future Work

In this dissertation, we have pursued a single goal: controllable image synthesis with high photorealism and low cost. Classical rendering and learned generative models approach photorealistic image synthesis in different ways. Classical rendering carries precise control and consistency, but bounded photorealism at a high authoring and compute cost; generative models learn photorealism at scale, but offer only a coarse control interface. We have argued that G-buffers are the right interface through which to combine the two: structured enough to carry the precise control of a rendering pipeline, and incomplete enough to leave a learned generative model room to synthesise the texture, imperfection, and high-frequency photorealism that is expensive to author and to simulate. We have built the resulting family of models, $\text{RGB} \leftrightarrow \text{X}$, as a pair: $\text{X} \rightarrow \text{RGB}$ renders photorealistic images from G-buffers, and $\text{RGB} \rightarrow \text{X}$ reads the same channels off a real image. Together, they address all three parts of the goal: control through the G-buffer interface, photorealism by learning directly from real data, and cost by shifting the burden of authoring and light transport from explicit scene assets and Monte-Carlo path tracing onto a learned prior that absorbs both.

In Chapter 3, we validated this argument in the image domain. A pair of latent-diffusion models fine-tuned from Stable Diffusion 2.1 operated on the same five-channel G-buffer set, with $\text{RGB} \rightarrow \text{X}$ using the text prompt as a per-channel modality switch and

$X \rightarrow \text{RGB}$ using per-channel dropout to train and test on arbitrary subsets of channels. Training drew on a heterogeneous mix of synthetic and real datasets, and $X \rightarrow \text{RGB}$ was augmented with a dataset of real photographs annotated by our own $\text{RGB} \rightarrow X$ model. The resulting system matched or improved on prior intrinsic-decomposition methods on every available channel and produced plausible, controllable forward renderings, including editing applications – material replacement and object insertion – that relied on both directions simultaneously. A natural extension is a richer authoring interface that moves away from dense per-pixel buffers: the segment-level spatial dropout of Section 4.3.4 hints at neural descriptors attached to coarse segmentations as a more tractable control primitive, and the five-channel G-buffer itself could be broadened with emissive, subsurface, or anisotropic parameters given the right training data.

In Chapter 4, we lifted the image-only, single-lighting, synthetic-data restrictions of Chapter 3 within a single framework, **RGBX-Next**. A diffusion-transformer video backbone replaced the U-Net, and a token-level finetuning recipe – frame-wise concatenation, QK type embedding, clean input tokens, and X-patchify – turned it into both a forward and an inverse generative renderer without modifying the architecture. We annotated roughly six thousand real Pexels videos with G-buffers using our own short-clip $\text{RGB} \rightarrow X$ model, captioned each clip with a vision-language model, and at inference used classifier-free guidance with positive “real” and negative “synthetic” prompts to push $X \rightarrow \text{RGB}$ toward the photorealistic mode of its training distribution. We also broadened the lighting interface with an albedo-free direct-lighting channel and multi-scale dropout, and extended both models to streaming variants via teacher forcing, long-context reference tokens, and a self-forcing finetune.

The main representation-level limitation is that G-buffers are validated here as a useful control language for generative rendering, not proven to be optimal. They carry geometry, color, material, and lighting information while leaving room for the learned

prior to synthesize photorealistic detail, which is precisely the balance exploited by both Chapter 3 and Chapter 4. Other control representations, including sparse segment descriptors, object-level material labels, explicit 3D features, learned lighting descriptors, or hybrid scene representations, may provide better authoring interfaces in some settings.

The streaming models of Section 4.4 are teacher-sized and inherit the per-frame cost of a 14B-parameter backbone, far from the frame rates expected of real-time rendering. Distillation methods such as DMD [37, 38], possibly combined with architecture changes [16, 41, 42], are the natural next step; whether they interact cleanly with the token-level modifications in this dissertation is an open question. The same streaming setup summarises past context through a single long-context reference keyframe, which prevents drift over hundreds of frames but is unlikely to suffice over minutes: multiple keyframes, image-space 3D feature grids, or explicit scene memories such as neural radiance fields or Gaussian splats are natural extensions. The lighting interface is another axis for future work. We have used two lighting representations, diffuse irradiance and albedo-free direct lighting, and argued that neither is universal; integrating HDR environment maps, image-space probe grids [36], learned lighting descriptors [43], or explicit light-source geometry into a single model, and working out when each should dominate, is a natural continuation. The same machinery points directly at relighting as a downstream application that sits alongside recent neural relighting work [44, 45, 46].

A second group of limitations concerns the training data. Section 4.3 trains $X \rightarrow RGB$ on roughly six thousand Pexels clips annotated by our own $RGB \rightarrow X$ model: the scale is small by modern video-pretraining standards, and the supervision is pseudo-ground-truth, so any systematic error in $RGB \rightarrow X$ is inherited by $X \rightarrow RGB$. Iterative self-training, curated paired real data from production sets or light-stage captures, and explicit uncertainty estimates on the $RGB \rightarrow X$ side are all natural ways to tighten this bottleneck. The captions themselves are another weak link: we rely on a general-purpose vision-language

model, and replacing the free-form text signal with richer structured metadata – scene graphs, camera intrinsics, object identities – would give the model a finer text channel than a single sentence per clip.

Another limitation applies to both Chapter 3 and Chapter 4. View consistency, geometric correctness, and temporal coherence are learned rather than enforced: neither framework has access to an explicit 3D scene, and the output pixels can drift under large camera motions or repeated passes over the same surface. Feeding explicit 3D inputs – meshes, Gaussian splats, or feed-forward reconstructions of the scene – through the same frame-wise concatenation interface is a natural way to reintroduce the consistency guarantees that a classical renderer already provides, complementary to the memory-oriented keyframe extensions above.

More broadly, our work points toward a deeper integration with the classical rendering pipeline, rather than a separate toolset alongside it. G-buffers are already the deferred-shading interface of real-time engines; $\mathbf{X} \rightarrow \mathbf{RGB}$ could, in principle, serve as a final shading stage that replaces or augments the physically based shader, while $\mathbf{RGB} \rightarrow \mathbf{X}$ remains a tool for analysing and editing real footage within the same pipeline. Making this practical requires progress on latency, memory, authoring, and lighting all at once, and would bring $\mathbf{RGB} \leftrightarrow \mathbf{X}$ to its natural conclusion: not an alternative to classical rendering, but a different way of completing it.

Taken together, our work suggests that structured physical control and learned photorealism are not competing answers to the same image-synthesis problem, but two halves of the same answer, with G-buffers as the piece of machinery that lets them fit. The hope, advanced here and left for future work to realise, is that image-synthesis systems built around this idea can offer artists and applications a tradeoff between precision, cost, and photorealism that neither classical rendering nor generative models can offer on their own.

Bibliography

- [1] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
- [2] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, *Microfacet models for refraction through rough surfaces*, EGSR’07, p. 195–206, 2007.
- [3] B. Burley and W. D. A. Studios, *Physically-based shading at disney*, in *Acm Siggraph*, vol. 2012, pp. 1–7, vol. 2012, 2012.
- [4] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, *Advances in neural information processing systems* **33** (2020) 6840–6851.
- [5] T. Salimans and J. Ho, *Progressive distillation for fast sampling of diffusion models*, *arXiv preprint arXiv:2202.00512* (2022).
- [6] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, *Flow matching for generative modeling*, *arXiv preprint arXiv:2210.02747* (2022).
- [7] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.
- [8] J. Ho and T. Salimans, *Classifier-free diffusion guidance*, *arXiv preprint arXiv:2207.12598* (2022).
- [9] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, *Lora: Low-rank adaptation of large language models*, *arXiv preprint arXiv:2106.09685* (2021).
- [10] L. Zhang, A. Rao, and M. Agrawala, *Adding conditional control to text-to-image diffusion models*, 2023.
- [11] W. Peebles and S. Xie, *Scalable diffusion models with transformers*, in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

- [12] T. Wan, A. Wang, B. Ai, B. Wen, C. Mao, C.-W. Xie, D. Chen, F. Yu, H. Zhao, J. Yang, J. Zeng, J. Wang, J. Zhang, J. Zhou, J. Wang, J. Chen, K. Zhu, K. Zhao, K. Yan, L. Huang, M. Feng, N. Zhang, P. Li, P. Wu, R. Chu, R. Feng, S. Zhang, S. Sun, T. Fang, T. Wang, T. Gui, T. Weng, T. Shen, W. Lin, W. Wang, W. Wang, W. Zhou, W. Wang, W. Shen, W. Yu, X. Shi, X. Huang, X. Xu, Y. Kou, Y. Lv, Y. Li, Y. Liu, Y. Wang, Y. Zhang, Y. Huang, Y. Li, Y. Wu, Y. Liu, Y. Pan, Y. Zheng, Y. Hong, Y. Shi, Y. Feng, Z. Jiang, Z. Han, Z.-F. Wu, and Z. Liu, *Wan: Open and advanced large-scale video generative models*, 2025.
- [13] W. Kong, Q. Tian, Z. Zhang, R. Min, Z. Dai, J. Zhou, J. Xiong, X. Li, B. Wu, J. Zhang, K. Wu, Q. Lin, J. Yuan, Y. Long, A. Wang, A. Wang, C. Li, D. Huang, F. Yang, H. Tan, H. Wang, J. Song, J. Bai, J. Wu, J. Xue, J. Wang, K. Wang, M. Liu, P. Li, S. Li, W. Wang, W. Yu, X. Deng, Y. Li, Y. Chen, Y. Cui, Y. Peng, Z. Yu, Z. He, Z. Xu, Z. Zhou, Z. Xu, Y. Tao, Q. Lu, S. Liu, D. Zhou, H. Wang, Y. Yang, D. Wang, Y. Liu, J. Jiang, and C. Zhong, *Hunyuanvideo: A systematic framework for large video generative models*, 2025.
- [14] N. Agarwal, A. Ali, M. Bala, Y. Balaji, E. Barker, T. Cai, P. Chattopadhyay, Y. Chen, Y. Cui, Y. Ding, *et. al.*, *Cosmos world foundation model platform for physical ai*, *arXiv preprint arXiv:2501.03575* (2025).
- [15] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, *Roformer: Enhanced transformer with rotary position embedding*, *Neurocomputing* **568** (2024) 127063.
- [16] X. Huang, Z. Li, G. He, M. Zhou, and E. Shechtman, *Self forcing: Bridging the train-test gap in autoregressive video diffusion*, *arXiv preprint arXiv:2506.08009* (2025).
- [17] L. Zhang, S. Cai, M. Li, G. Wetzstein, and M. Agrawala, *Frame context packing and drift prevention in next-frame-prediction video diffusion models*, in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.
- [18] Z. Zeng, V. Deschaintre, I. Georgiev, Y. Hold-Geoffroy, Y. Hu, F. Luan, L.-Q. Yan, and M. Hašan, *Rgb $\leftrightarrow$ x: Image decomposition and synthesis using material- and lighting-aware diffusion models*, in *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24, (New York, NY, USA), Association for Computing Machinery, 2024.
- [19] J. Zhu, F. Luan, Y. Huo, Z. Lin, Z. Zhong, D. Xi, R. Wang, H. Bao, J. Zheng, and R. Tang, *Learning-based inverse rendering of complex indoor scenes with differentiable monte carlo raytracing*, in *SIGGRAPH Asia 2022 Conference Papers*, ACM, 2022.
- [20] M. Roberts, J. Ramapuram, A. Ranjan, A. Kumar, M. A. Bautista, N. Paczan, R. Webb, and J. M. Susskind, *Hypersim: A photorealistic synthetic dataset for*

holistic indoor scene understanding, in *International Conference on Computer Vision (ICCV) 2021*, 2021.

- [21] NVIDIA, *Nvidia optixTM ai-accelerated denoiser*, 2020.
- [22] J. Li, D. Li, S. Savarese, and S. Hoi, *Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models*, 2023.
- [23] P. Kocsis, V. Sitzmann, and M. Nießner, *Intrinsic image diffusion for single-view material estimation*, in *arxiv*, 2023.
- [24] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et. al.*, *Learning transferable visual models from natural language supervision*, in *International conference on machine learning*, pp. 8748–8763, PMLR, 2021.
- [25] T. Brooks, A. Holynski, and A. A. Efros, *Instructpix2pix: Learning to follow image editing instructions*, in *CVPR*, 2023.
- [26] P. Sharma, V. Jampani, Y. Li, X. Jia, D. Lagun, F. Durand, W. T. Freeman, and M. Matthews, *Alchemist: Parametric control of material properties with diffusion models*, 2023.
- [27] L. Huang, D. Chen, Y. Liu, Y. Shen, D. Zhao, and J. Zhou, *Composer: Creative and controllable image synthesis with composable conditions*, *arXiv preprint arXiv:2302.09778* (2023).
- [28] C. Careaga and Y. Aksoy, *Intrinsic image decomposition via ordinal shading*, *ACM Trans. Graph.* (2023).
- [29] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, *Pvt v2: Improved baselines with pyramid vision transformer*, *Computational Visual Media* **8** (2022), no. 3 415–424.
- [30] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, *Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **44** (2022), no. 3.
- [31] R. Birkel, D. Wofk, and M. Müller, *Midas v3.1 – a model zoo for robust monocular relative depth estimation*, *arXiv preprint arXiv:2307.14460* (2023).
- [32] Jay-Artist, *Country-kitchen cycles*, 2012.
- [33] Z. Jiang, Z. Han, C. Mao, J. Zhang, Y. Pan, and Y. Liu, *Vace: All-in-one video creation and editing*, 2025.

- [34] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie, Z. Cheng, H. Zhang, Z. Yang, H. Xu, and J. Lin, *Qwen2.5-vl technical report, arXiv preprint arXiv:2502.13923* (2025).
- [35] R. Liang, Z. Gojcic, H. Ling, J. Munkberg, J. Hasselgren, Z.-H. Lin, J. Gao, A. Keller, N. Vijaykumar, S. Fidler, *et. al.*, *Diffusionrenderer: Neural inverse and forward rendering with video diffusion models, arXiv preprint arXiv:2501.18590* (2025).
- [36] Z. Li, J. Shi, S. Bi, R. Zhu, K. Sunkavalli, M. Hašan, Z. Xu, R. Ramamoorthi, and M. Chandraker, *Physically-based editing of indoor scene lighting from a single image*, in *ECCV 2022*, pp. 555–572, 2022.
- [37] T. Yin, M. Gharbi, R. Zhang, E. Shechtman, F. Durand, W. T. Freeman, and T. Park, *One-step diffusion with distribution matching distillation*, in *CVPR*, 2024.
- [38] T. Yin, M. Gharbi, T. Park, R. Zhang, E. Shechtman, F. Durand, and W. T. Freeman, *Improved distribution matching distillation for fast image synthesis*, in *NeurIPS*, 2024.
- [39] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium, Advances in neural information processing systems* **30** (2017).
- [40] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, *et. al.*, *Sam 2: Segment anything in images and videos, arXiv preprint arXiv:2408.00714* (2024).
- [41] S. Lin, C. Yang, H. He, J. Jiang, Y. Ren, X. Xia, Y. Zhao, X. Xiao, and L. Jiang, *Autoregressive adversarial post-training for real-time interactive video generation*, 2025.
- [42] S. Yang, W. Huang, R. Chu, Y. Xiao, Y. Zhao, X. Wang, M. Li, E. Xie, Y. Chen, Y. Lu, and S. H. Y. Chen, *Longlive: Real-time interactive long video generation*, .
- [43] P. Kocsis, J. Philip, K. Sunkavalli, M. Nießner, and Y. Hold-Geoffroy, *Lightit: Illumination modeling and control for diffusion models*, in *CVPR*, 2024.
- [44] H. Jin, Y. Li, F. Luan, Y. Xiangli, S. Bi, K. Zhang, Z. Xu, J. Sun, and N. Snavely, *Neural gaffer: Relighting any object via diffusion*, in *Advances in Neural Information Processing Systems*, 2024.
- [45] L. Zhang, A. Rao, and M. Agrawala, *Scaling in-the-wild training for diffusion-based illumination harmonization and editing by imposing consistent light transport*, in *The Thirteenth International Conference on Learning Representations*, 2025.

- [46] K. He, R. Liang, J. Munkberg, J. Hasselgren, N. Vijaykumar, A. Keller, S. Fidler, I. Gilitschenski, Z. Gojcic, and Z. Wang, *Unirelight: Learning joint decomposition and synthesis for video relighting*, *arXiv preprint arXiv:2506.15673* (2025).