GFFE: G-buffer Free Frame Extrapolation for Low-latency Real-time Rendering

SONGYIN WU, University of California Santa Barbara, USA
DEEPAK VEMBAR, Intel Corporation, USA
ANTON SOCHENOV, Intel Corporation, USA
SELVAKUMAR PANNEER, Intel Corporation, USA
SUNGYE KIM, Intel Corporation (now AMD), USA
ANTON KAPLANYAN, Intel Corporation, USA
LING-QI YAN, University of California Santa Barbara, USA

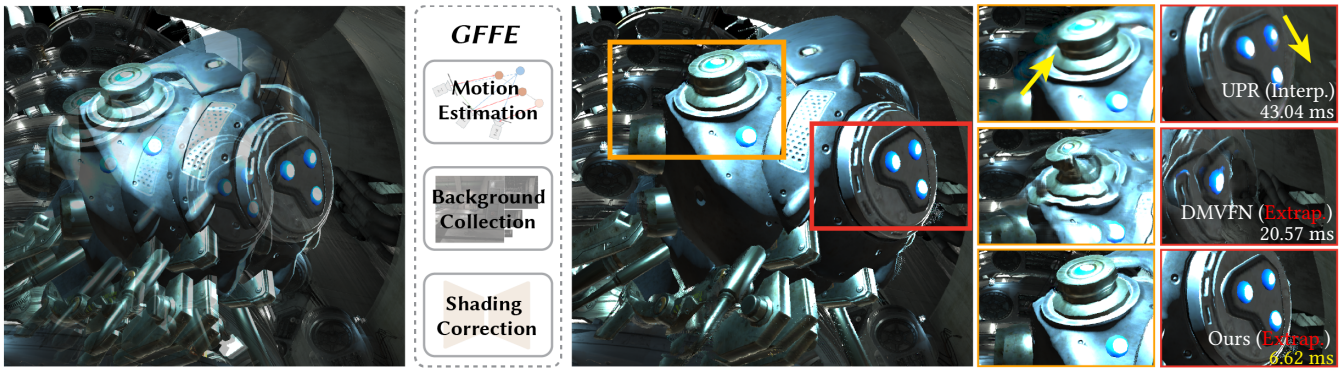

Fig. 1. We propose a *G-buffer free* frame extrapolation framework, GFFE, which introduces no additional latency (unlike interpolation methods) and eliminates the need for additional G-buffer generation of extrapolated frames. Our framework shows better visual quality than the previous frame extrapolation method DMVFN [Hu et al. 2023] and interpolation method UPR [Jin et al. 2023] with better performance.

Real-time rendering has been embracing ever-demanding effects, such as ray tracing. However, rendering such effects in high resolution and high frame rate remains challenging. Frame extrapolation methods, which do not introduce additional latency as opposed to frame interpolation methods such as DLSS 3 and FSR 3, boost the frame rate by generating future frames based on previous frames. However, it is a more challenging task because of the lack of information in the disocclusion regions and complex future motions, and recent methods also have a high engine integration cost due to requiring G-buffers as input. We propose a *G-buffer free* frame extrapolation method, GFFE, with a novel heuristic framework and an efficient neural network, to plausibly generate new frames in real time without introducing additional latency. We analyze the motion of dynamic fragments and different types of disocclusions, and design the corresponding modules of the extrapolation block to handle them. After that, a light-weight shading correction network is used to correct shading and improve overall quality. GFFE achieves comparable or better results than previous interpolation and G-buffer dependent extrapolation methods, with more efficient performance and easier integration.

CCS Concepts: • **Computing methodologies** → **Rendering**.

Additional Key Words and Phrases: Extrapolation, Low Latency, Warping, G-buffer Free

ACM Reference Format:
Songyin Wu, Deepak Vembar, Anton Sochenov, Selvakumar Panneer, Sungye Kim, Anton Kaplanyan, and Ling-Qi Yan. 2024. GFFE: G-buffer Free Frame Extrapolation for Low-latency Real-time Rendering. *ACM Trans. Graph.* 43, 6, Article 248 (December 2024), 15 pages. https://doi.org/10.1145/3687923

1 Introduction

Real-time rendering has advanced significantly in recent years to create more realistic and interactive environments, including the recent trend for real-time path tracing effects in games. Usually, high quality and high frame rates are required for games or virtual reality applications in order to provide a good user experience. However, the cost of rendering such high quality frames is expensive even for the most powerful graphics hardware - naively rendering all frames is not always possible under fixed computing and power budgets. Therefore, in addition to methods that accelerate frame rendering,

Authors' Contact Information: Songyin Wu, s_wu975@ucsb.edu, University of California Santa Barbara, USA; Deepak Vembar, dvembar@gmail.com, Intel Corporation, USA; Anton Sochenov, anton.sochenov@intel.com, Intel Corporation, USA; Selvakumar Panneer, selvakumar.panneer@intel.com, Intel Corporation, USA; Sungye Kim, sungyekim@gmail.com, Intel Corporation (now AMD), USA; Anton Kaplanyan, anton.kaplanyan@intel.com, Intel Corporation, USA; Ling-Qi Yan, lingqi@cs.ucsb.edu, University of California Santa Barbara, USA.

ACM 1557-7368/2024/12-ART248
https://doi.org/10.1145/3687923

approaches such as frame super resolution and generation [AMD 2021; Guo et al. 2021, 2022; Intel 2022; Liu 2020; NVIDIA 2022; Wu et al. 2023a,b; Xiao et al. 2020] are usually implemented in a separated post-processing pass to provide the best quality output within given compute budgets.

Frame generation is one technique that can be used to increase the frame rate for a smoother and jitter-free experience. Frame interpolation, including proprietary products DLSS 3 [NVIDIA 2022], FSR 3 [AMD 2022] and research works [Briedis et al. 2021, 2023; Jin et al. 2023; Kong et al. 2022; Yang et al. 2024] try to generate new frames between two rendered frames. These methods increase the key-press-to-display latency of the rendering process since the generated frames rely on the availability of both the previous and the next frames.

Frame extrapolation, on the other hand, generates new frames based solely on previous frames, and does not introduce additional latency to the rendering process. However, it is a more difficult task and usually generates inferior results due to the missing information from the future frames. Many existing methods, including ExtraNet [Guo et al. 2021], LMV [Wu et al. 2023b], and ExtraSS [Wu et al. 2023a], use G-buffers of target frames to guide the generation of corresponding final frames. Game-generated G-buffers are not always easily available and the cost of obtaining them from various rendering pipelines is not negligible. Other video extrapolation methods [Hu et al. 2023] do not require G-buffers to generate color frames, but they usually have inferior quality and performance under real-time rendering settings.

Existing methods have shown abilities to generate new frames, but they either introduce latency or require additional G-buffers. Motivated by these problems, we propose a novel method to generate new frames without introducing latency or requiring additional G-buffers. Our insight is the missing information of extrapolated frames can be approximately retrieved from previous frames, which are usually discarded in the rendering pipeline. Additionally, the motion of fragments can be plausibly estimated from history frames, so there is no need to render G-buffers for extrapolated frames.

Based on these observations, we propose a G-buffer free extrapolation framework. First, it uses a heuristic motion estimation method to eliminate the requirement of rendering motion vectors for extrapolated frames. Then, to handle disocclusions in the extrapolated frames, we introduce a background collection module and adaptive rendering window. Lastly, we use a light-weight neural network to improve the shading and shadow consistency further.

To demonstrate our quality, performance, and robustness, we evaluate our framework on various scenes in Unreal Engine [Epic Games 2022] with different effects, including glossy and translucent materials, complex geometry, and dynamic objects. Our method generates smooth and plausible results from 30 FPS to 60 FPS. It shows superior quality to G-buffer free extrapolation baseline and comparable results with G-buffer dependent baseline and interpolation baselines, with better visual quality and performance.

2 Related Work

2.1 Warping and Hole Filling

Warping has been used in real-time rendering for many years to improve quality and performance. Mark et al. [1997] propose a 3D warping method to warp the frame to new frames as a post-processing step. However, it is difficult for disoccluded areas since such information is not available, and a hole filling algorithm is needed. Didyk et al. [2010] employ additional blur operations to the warped frames to reduce the artifacts for disocclusion areas. Similar to Didyk et al. [2010], Schollmeyer et al. [2017] propose a hole filling method to fill the disocclusion areas by low pass filtering of them to reduce the artifacts. These methods bring blurry artifacts to the extrapolated frames instead of generating actual details, which is unsuitable for modern real-time rendering. Later, Reinert et al. [2016] build geometry proxies to fill the disocclusion areas but require pre-computed geometry information and still in low quality since they use low poly geometries. Wu et al. [2023a] and Zeng et al. [2021] use G-buffers to guide the hole filling process by reusing spatial neighbors' information. However, the G-buffers are not always available in real-time rendering, which limits the usage of these methods.

Besides single frame warping methods which cannot retrieve valid information in disocclusions, there are also some bidirectional methods trying to warp frames from both previous and future frames. Andreev [2010] uses half motion vectors to warp both the previous and future frame to the current frame to increase the frame rate. Yang et al. [2011] use an iterative way to find the correspondence from previous and future frames to the current frame. Although these methods fill the disocclusion areas better, they introduce additional key-press-to-display latency since new frames rely on future frames, and usually, the quality is not good enough including lagging shadows and shadings.

2.2 Frame Interpolation

Besides pure warping based methods, several frame interpolation methods with neural networks achieve better quality. Briedis et al. [2021, 2023] propose using optical flows or kernel prediction neural network to generate intermediate frames from given corresponding G-buffers. Although the quality looks promising, these techniques are used for offline rendering, which is inefficient in real-time rendering due to low performance. Video interpolation methods [Bao et al. 2019; Huang et al. 2022; Jin et al. 2023; Kong et al. 2022] also generate plausible intermediate frames with neural networks but usually with blurrier results and worse performance since these methods are not designed for the rendering pipeline. Commercial solutions including DLSS 3 [NVIDIA 2022] are also proposed to boost frame rate in games but the details of their methods are not released and usually require specific hardware. Offline frame interpolation methods [Reda et al. 2022; Zhang et al. 2023; Zhou et al. 2023] take more than 100 ms per frame, which is impractical in real-time rendering settings. Nevertheless, frame interpolation methods introduce additional latency, making users feel lagging when interacting with the scene. Such latency becomes more severe when the input frame rate is low, such as boosting 30 FPS to 60 FPS.

Table 1. Features and challenges of different frame generation methods.

	G-buf Free Interp.	G-buf Dependent Extrap.	G-buf Free Extrap.
Low latency		✓	✓
No-extra G-buffers	✓		✓
Motion Est.	✓		✓
Disocclusion		✓	✓
Non-geo Tracking	✓	✓	✓

Fig. 2. An extrapolated frame by directly projecting fragments from the previously rendered frame. The right column shows three types of disocclusions: *out-of-screen disocclusion*, *static disocclusion*, and *dynamic disocclusion* from top to bottom. The thin black lines splatted in the image are due to forward warping.

2.3 Frame Extrapolation

To avoid the extra latency introduced by frame interpolation while increasing frame rate, frame extrapolation methods have been explored these years to generate new frames only based on history frames. ExtraNet [Guo et al. 2021] uses occlusion motion vectors [Zeng et al. 2021] with a neural network to handle both disocclusions and lagging shadings. Learnable motion vector [Wu et al. 2023b] proposes a recurrent framework to optimize motion vectors to handle the motion of shadings and disocclusion areas. ExtraSS [Wu et al. 2023a] uses G-buffers to guide the extrapolation process and a flow-based neural network to fix the shading errors. These methods require G-buffers for extrapolated frames, which is not always the case in real-time rendering from different engines and platforms such as mobiles, cloud gaming, and some forward rendering engines. Concurrent work [Yang et al. 2024] uses simple warping and hole filling methods for extrapolation but fails with large disocclusions and does not consider shading's motion.

Video extrapolation methods [Hu et al. 2023], although they do not require G-buffers for extrapolated frames, usually yield much worse quality and performance, which are unsuitable for real-time rendering. Li et al. [2022] use optical flow to predict future frames, but a re-shading process is needed to refine extrapolated frames, which differs from our settings.

3 Motivation

3.1 Problem Formulation and Design Choices

Our G-buffer free extrapolation framework aims to extrapolate new frames without dependence on G-buffers for extrapolated frames and additional latency. Note that we use the term "G-buffer free" to refer to the absence of G-buffers for extrapolated frames only. The depth buffer and motion vectors for rendered frames are used since they are usually readily available in the rendering engine without additional cost. Unlike previous G-buffer based extrapolation methods [Wu et al. 2023a,b], we do not use G-buffers for extrapolated frames as well as some types of G-buffers, including albedo, normal, and roughness for rendered frames.

We formulate our problem as follows: given a sequence of rendered frames $\{I_t\}$ with their corresponding depth buffer $\{D_t\}$ and motion vectors $\{V_t\}$, our framework generates new frames $\{\bar{I}_{t+\alpha}\}$ with their corresponding depth buffer $\{\bar{D}_{t+\alpha}\}$ and motion vectors $\{\bar{V}_{t+\alpha}\}$, where α depends on the number of frames we want to generate for every rendered frame.

In addition to our G-buffer free frame extrapolation, two other types of methods are commonly used: G-buffer free frame **interpolation** and G-buffer **dependent** frame extrapolation. The features of these three types of methods are shown in Table 1.

Latency. Frame interpolation methods are widely used and have demonstrated good quality as it is easier to find correspondence in either previous or latter frames. The main disadvantage of interpolation methods is the additional latency introduced. As analyzed in previous works [Guo et al. 2021; Wu et al. 2023a,b], the latency of interpolation methods is increased by at least one rendering time interval, which is even higher than the original latency without the frame interpolation method. This leads to a worse user experience especially when low latency is required such as in competitive games [Kim et al. 2020; Spjut et al. 2019, 2021] and virtual reality applications [van Waveren 2016]. Some works [Kim et al. 2020; Oculus 2016; van Waveren 2016] attempt to decrease the latency for a better experience, while frame generation methods introduce extra latency. Although mitigation techniques such as NVIDIA Reflex can be used to decrease the latency from the frame interpolation, they still cannot fully eliminate it.

G-buffers. To avoid introducing additional latency, frame extrapolation methods have been proposed [Guo et al. 2021; Wu et al. 2023a,b]. Since achieving similar quality compared to frame interpolation methods is more challenging, various types of G-buffers from extrapolated frames are required. However, it is not always practical due to the different engine types (forward rendering engines) and G-buffer generation cost. More discussion about this is included in the appendix.

Besides, we also consider our G-buffer free frame extrapolation framework for possible future applications, especially low-latency streaming and cloud gaming on various low end devices. In order to provide an immediate response to the user inputs, the frame extrapolation should be done on the client side, where scene information is not available to generate G-buffers. Therefore, our G-buffer free frame extrapolation framework is more suitable for such applications than frame interpolation or G-buffer dependent extrapolation.

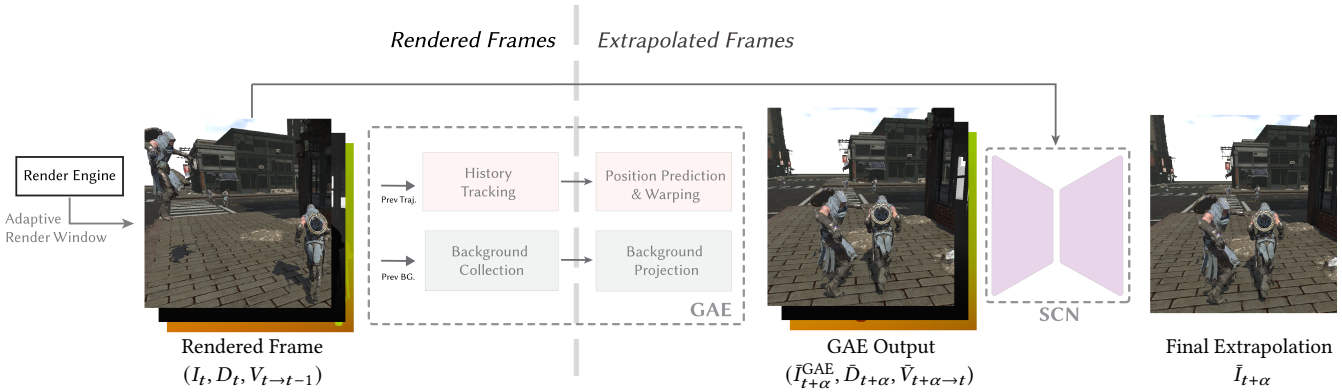

Fig. 3. Our method generates an extrapolated frame $\bar{I}_{t+\alpha}$ from the rendered frame I_t and history frames. The left part shows the process of rendered frames including adaptive rendering window, history tracking, and background collection, which are prepared for extrapolated frames. The right part shows the process of extrapolating a frame, including geometry aware extrapolation (GAE) and shading correction network (SCN). The depth and motion vectors in extrapolated frames are generated in our framework instead of the rendering engine, which can be used for additional post-processing.

3.2 Challenges

3.2.1 Motion estimation. Our method works under the assumption that the rendering engine does not generate any G-buffers for extrapolated frames. Therefore, unlike previous G-buffer dependent extrapolation methods [Guo et al. 2021; Wu et al. 2023a,b], the motion from rendered frame I_t to extrapolated frame $\bar{I}_{t+\alpha}$ is unknown. Previous warping methods [Bowles et al. 2012; Lee et al. 2018; Mark et al. 1997] work either only on static scenes or where the objects' motion is given, so no motion estimation is needed.

Motion estimation is challenging since the motion of the objects in the game can be arbitrarily complex. Frame interpolation methods [Jin et al. 2023; Kong et al. 2022] use neural networks to predict the motion between two rendered frames, but they are usually slow and sometimes unstable.

Instead of using heavy and slow neural networks to predict motion, we use a heuristic motion approximation method to estimate the motion for each dynamic fragment for extrapolated frames. Our goal is to estimate plausible motion in order to achieve smooth transitions in continuous frames - we do not expect perfectly estimated motion since the future frames' motion can be arbitrary.

3.2.2 Disocclusions. Disocclusions, as shown in Fig. 2, are areas that are not shown in the previous frame but visible in the current frame. They are challenging to handle with the frame extrapolation approach due to the lack of information from the next rendered frame, which is used in frame interpolation frameworks. Prior frame extrapolation methods use G-buffers for the extrapolated frame. Although G-buffers are not shaded, they provide sufficient information to fill disocclusions. However, under our settings, there is no such information in either the previous frame or G-buffers, which makes it more difficult to recover this information.

To better understand and deal with the disocclusions, we categorize them into three types: **(1) Out-of-screen disocclusion:** Pixels on the boundaries shown in the current frame but not in the screen space of the previous frame. This disoccluded area is caused by the camera's motion and happens in the image boundaries. **(2) Static disocclusion:** Pixels that are shown in the current frame but not in the previous frame due to occlusion from static occluders. These pixels are static in the two consecutive frames and in the screen space of the previous frames. They become visible in the current frame due to the change of the camera's position. **(3) Dynamic disocclusion:** Similar to the static disocclusion, the only difference is that the occluders are dynamic. These areas are usually caused by the motion of the occluders instead of the camera.

Simply using a neural network to fill the disocclusions causes artifacts as it does not have any information in those areas, and the size[1] of the disocclusions areas, which are usually not small, as shown in Fig. 2. Our proposed method uses history information with efficient adaptive rendering windows to handle the disocclusions more plausibly.

3.2.3 Non-geometric motion tracking. Frame generation methods usually reuse temporal information and find corresponding pixels in existing frames. However, such correspondence computation is not always accurate. The color in the rendered frames is the combination of lighting information with materials' properties, which may have different directions of motion. Rendered motion vectors only capture geometries' motion but not the lightings' motion. Only considering the geometries' motion like Mob-FGSR [Yang et al. 2024] causes lagging in shading and shadows [Guo et al. 2021; Wu et al. 2023a,b]. For example, shadows move at 30 FPS while other objects move at 60 FPS. Although shadows and reflections contribute a small portion to common metrics such as PSNR and SSIM, they are crucial for visual quality. Therefore, a module for tracking such motion is necessary to maintain a high frame rate in all areas with a smooth transition between frames. To address this, we design our shading correction network to fix these issues.

[1]The size of disocclusions depends on the frame rate and objects' motion speed. We target 30 FPS inputs which usually have noticeable disocclusions

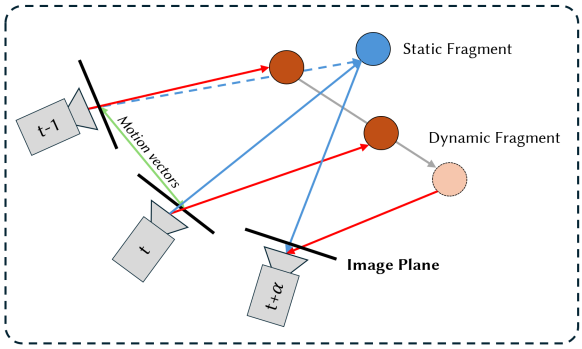

Fig. 4. Our motion estimation module tracks history trajectories and estimates the next positions in world space based on history trajectory.

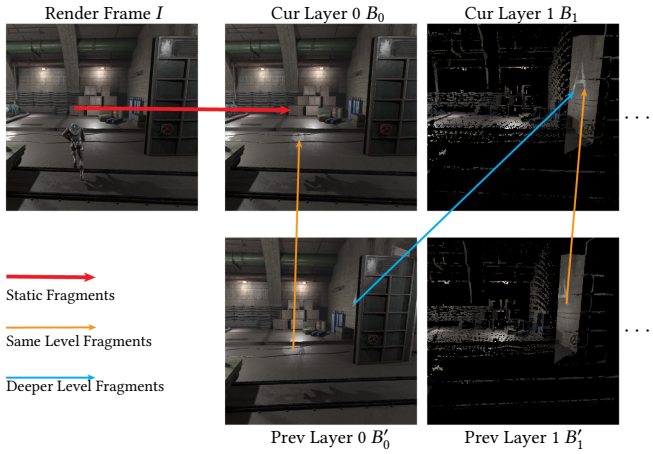

Fig. 5. The process of layered background collection. The top row is a current rendered frame and updated background buffers, and the bottom row is the previous background buffers. Different color arrows show different conditions when updating the background buffers. The size of deeper layers (Layer 1) is only 1/4 than the previous layer (Layer 0).

4 Method

The overview of our framework is shown in Fig. 3. Challenges mentioned in Sec. 3.2 are addressed by our different modules: motion estimation (Sec. 4.1), background collection for static and dynamic disocclusions (Sec. 4.2), adaptive rendering window for out-of-screen disocclusions (Sec. 4.3) and shading correction network for non-geometric motion tracking (Sec. 4.4). We detail each component in the following sections.

4.1 Motion Estimation

Frame extrapolation usually re-uses history frame information to generate new frames. Existing extrapolation methods [Guo et al. 2021; Wu et al. 2023a,b] use motion vectors from rendering engines to find the corresponding pixels in the previous frame. These motion vectors are accurate but require a rasterization process for extrapolated frames. We propose a motion estimation module to efficiently predict the motion of fragments for extrapolated frames.

The motion estimation module consists of three parts: history tracking, position estimation, and warping. It first collects the history trajectory in world space and uses it to estimate the next positions in world space for extrapolated frames. After that, a warping process is applied to warp fragments to extrapolated frames.

History tracking. History tracking happens in rendered frames, which calculates the history trajectory of each fragment in world space. At a high level, our history tracking algorithm works recurrently for each rendered frame to generate k history world positions $\{P_i[x]\}$ for each pixel, where the current trajectory is updated from the corresponding previous trajectory. In order to avoid incorrect correspondences due to disocclusions, we designed a static test algorithm by comparing the previous screen space of each pixel by using motion vectors and view projection matrices. If the distance is smaller than a threshold, the pixel is set as a static pixel, to avoid calculating incorrect history trajectory. The algorithm details are shown in the appendix in Algo. 1.

Position estimation. History tracking provides history world positions $\{P_i[x]\}$ for each pixel. For extrapolated frames, let α be the extrapolation factor which is calculated by $\alpha = \frac{j}{n+1}$, where n is the number of extrapolated frames per rendered frame, and j refers to the j-th extrapolated frame for a rendered frame. As shown in Fig. 4, the next positions $NP_{t\to t+\alpha}$ is estimated by calculating the linear motion of the last two positions in the trajectory

$$NP_{t\to t+\alpha}[x] = \alpha(P_0[x] - P_1[x]) + P_0[x] \tag{1}$$

Unlike calculating motion in the image space where linear motions are not reliable due to perspective projection, camera rotation, etc., the linear motion assumption in world space efficiently generates plausible next positions in world space. High-order polynomials could be used here, but they lead to worse results. Please refer to ablation studies for more analysis.

Warping. After calculating the next positions in world space, each fragment is projected to the extrapolated frame based on the camera view projection matrix. For multiple fragments projected into the same pixel, we compare the depth value for the projected pixels and keep the fragment with the smallest depth value using atomic operations. Although there are several works [Bowles et al. 2012; Lee et al. 2018] with better ways of warping/projection, our warping method is efficient and simple, which is already sufficient for our pipeline.

4.2 Layered Background Collection

With estimated motion and warping in Sec. 4.1, an initial extrapolated frame is generated but with invalid regions caused by disocclusions as analyzed in Sec. 3.2.2, where we use two modules to handle them. One insight is that static and dynamic disocclusions, although invisible in the previous frame, may show up in long history frames before. However, naively storing more history frames is impractical due to memory limits, and matching the correspondences is also time-consuming. Inspired by this, we propose a layered background collection module to collect useful information from history frames to fill disocclusions efficiently.

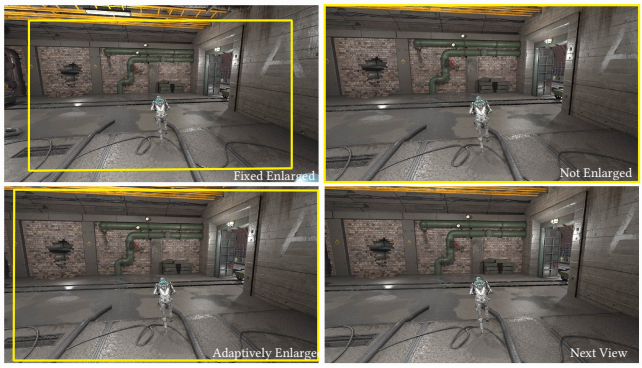

Fig. 6. Rendered image under different settings. The yellow rectangle shows displayed areas of the frame. All frames are rendered under the same resolution. Our adaptive strategy not only covers the area we need for the next view, but also contains less redundant information.

This module contains two parts: a background collection module to maintain a background buffer $\{B_l\}$ of collecting the fragments of rendered frames as well as the fragments behind the directly visible fragments without additional rendering cost, and a background projection process for extrapolated frames to fill disocclusions.

Background collection. Fig. 5 shows the process of background collection. The background buffer B contains L levels with a pair of a color buffer and a depth buffer for each level, denoting as $B = \{B_l\}$, and the size of the deeper level is only 1/4 to the previous level. Let (I_t, D_t) be the current rendered frame, and $B' = \{B'_l\}$ be the previous background buffer. The static fragments of rendered frame (I_t, D_t) are filled into the first layer of updated background buffer B_0. For each level l in the previously collected background B'_l, there two conditions to update the current background buffer B_l:

- **Case 1 (same level fragments):** If the corresponding position in the same level $B_l[x']$ is invalid, $B'_l[x]$ is used for filling $B_l[x']$.
- **Case 2 (deeper level fragments):** If the corresponding position in the same level $B_l[x']$ is already valid, and the depth value of $B'_l[x]$ is larger, $B'_l[x]$ is used for next level $B_{l+1}[x']$, meaning the deeper fragments of the current layer. If multiple fragments are projected into the same pixel, we keep the fragment with the smallest depth value that satisfies the condition.

Each level represents a layer of geometries in the scene, and the higher level contains the deeper fragments. Hence, we can keep track of the occluded fragments by updating the background from level 0 to L.

Background projection. For extrapolated frames, the collected background buffers are projected to the world space and then back to the extrapolated frames. We only fill the invalid regions of disocclusions in the extrapolated frames.

4.3 Adaptive Rendering Window

Out-of-screen disocclusion, unlike regions that can be handled by the background collection, is usually never shown in history frames such as continuously rotating cameras to the right. These disocclusion areas are on the boundary of the frame, and enlarging the original rendering viewport could cover them.

A naive way to solve it is to enlarge the field-of-view angle for rendered frames, but this includes redundant information, leading to blurry results for displayed areas at the same rendering cost. Instead, we propose an adaptive rendering window strategy to decrease the area of redundant regions as shown in Fig. 6.

Specifically, when rendering a frame t, we estimate the potential areas that will be used for extrapolated frames by two steps: estimate the next camera pose and calculate the rendering viewport. Assume camera pose of current frame is C_t and previous rendered frame is C_{t-1}, where the pose is formed by three vectors $(v^{\text{pos}}, v^{\text{dir}}, v^{up})$. To estimate the camera pose of extrapolated frames $\bar{C}_{t+\alpha}$, we use a similar method as our motion estimation by

$$\bar{C}_{t+\alpha} = C_t + \alpha \cdot (C_t - C_{t-1}) \tag{2}$$

where it calculates each vector component separately.

After calculating the estimated camera pose in the next extrapolated frame, the new rendering viewport is approximated based on the union of current camera pose C_t and estimated camera pose $\bar{C}_{t+\alpha}$ rendering areas, which is used for actual rendering. Please refer to the appendix for details on calculating the actual rendering viewport.

4.4 Shading Correction Network

Previous modules handle the motion of geometries, so we call them geometry aware extrapolation (GAE) modules. However, the motion of shadings is not tracked, and simply ignoring it causes shadings to move at a low frame rate as analyzed in Sec. 3.2. Thanks to our previous efficient modules which handle geometries motion and disocclusions, we introduce a light-weight neural network called shading correction network (SCN) for non-geometric motion tracking and refinement, which is unlike prior works UPR-Net [Jin et al. 2023], IFR-Net [Kong et al. 2022], and DMVFN [Hu et al. 2023] using large neural networks to estimate the flow for the whole image.

Non-geometric motion detection. To make SCN only focus on the non-geometric motion and shadings, we generate a focus mask to identify the areas that need to be refined and exclude the already plausible areas. The ground truth focus mask is calculated by the following formula:

$$\begin{aligned} M^{\text{focus}}[x] = &\left(\min_{x' \in N(x)} s(I^{\text{GAE}}[x], I^{\text{gt}}[x']) > 0.5 \right) \\ &\wedge (\hat{M}^{\text{dyn}}[x] = 0) \end{aligned} \tag{3}$$

where $s(\cdot,\cdot)$ refers to symmetric mean absolute percentage error (SMAPE), $N(x)$ is the set of 9 neighborhood pixels and $\hat{M}^{\text{dyn}}$ represents whether a pixel is dynamic. This mask will ignore subtle differences and pixels shifting to only focus on the shading changes.

Shading correction network. After calculating the focus mask, it guides the shading correction network to only focus on the non-geometric motion. Specifically, the inputs of the network contain: the output of GAE module $\bar{I}^{\text{GAE}}_{t+\alpha}$, the corresponding projected depth buffer $\bar{D}_{t+\alpha}$, the warped frame $I^{w}_{t-1\to t+\alpha}$ from rendered frame $t-1$

Fig. 7. The inputs of shading correction network (SCN). Images from left to right are: the output of GAE module $\bar{I}^{\text{GAE}}_{t+\alpha}$, the backward warped result $I^{w}_{t-1\to t+\alpha}$ from frame $t-1$ using motion vectors, the generated depth buffer $\bar{D}_{t+\alpha}$, and the input mask M^{input}_{t}.

Table 2. Scene configuration for training and testing. All frames are captured in 1080p/30FPS for inputs and 1080p/60FPS for outputs. Our dataset contains less training data and more diverse testing data compared to previous works [Guo et al. 2021; Wu et al. 2023a,b].

Scenes	Training Sequences	Testing Sequences	Training Frames	Testing Frames
Bunker	2	1	2000	720
Park	2	1	2000	720
Future	2	1	2000	720
City	2	1	2000	720
Town	0	1	0	720
Forest	0	1	0	720
Factory	0	1	0	720
Infiltrator	0	1	0	720

to $t+\alpha$ where ghosting areas are replaced with the corresponding areas in $\bar{I}^{\text{GAE}}_{t+\alpha}$, and the input mask M_{input}. An example of the shading correction network's inputs is shown in Fig. 7.

The input mask is generated by our GAE module, where the white region indicates dynamic areas, the black region indicates disocclusion areas, and the grey region indicates remaining areas. Warped frame from $t-1$ provides a different shading condition compared to $\bar{I}^{\text{GAE}}_{t+\alpha}$ in order to calculate the motion of the shading. The remaining invalid areas in $\bar{I}^{\text{GAE}}_{t+\alpha}$ will be filled by down-sampling the original image 32 times before feeding it into the network. The final prediction $\bar{I}_{t+\alpha}$ is formulated as

$$\begin{aligned} \bar{I}'_{t+\alpha},\ \bar{M}^{\text{focus}} &= \text{SCN}(\bar{I}^{\text{GAE}}_{t+\alpha}, \bar{D}_{t+\alpha}, I^{w}_{t-1\to t+\alpha}, M^{\text{input}}_{t}) \\ \bar{I}_{t+\alpha} &= \bar{I}^{\text{GAE}}_{t+\alpha} \cdot (1-\bar{M}^{\text{focus}}) + \bar{I}'_{t+\alpha} \cdot \bar{M}_{\text{focus}} \end{aligned} \tag{4}$$

where $\bar{I}^{\text{GAE}}_{t+\alpha}$ in the second formula is replaced by the ground truth frame $I_{t+\alpha}$ during training. After the SCN module, the extrapolated images not only contain correct geometries including dynamic fragments and disocclusions, but also correct shading movement. Please refer to the appendix for the detailed network architecture, loss functions, and training process.

5 Experiments

5.1 Datasets

To demonstrate the effectiveness and robustness of our method, we collect eight different scenes with different characteristics from Unreal Engine [Epic Games 2022]. To demonstrate the generalization ability of our method, we use four scenes for training and test on all scenes, where four scenes are never shown during training. The dataset details are shown in Table 2. Our collected test scenes are more diverse than previous works [Guo et al. 2021; Wu et al. 2023a,b] with less training data to show our robustness and generalization ability.

5.2 Quantitative Metrics

We evaluate our method with both quantitative metrics and qualitative images/videos to show the comparison. Four metrics are included to show various aspects of our quality: peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), perceptual similarity (LPIPS) [Zhang et al. 2018], and FovVideoVDP (FvVDP) [Mantiuk et al. 2021]. However, we noticed that PNSR and SSIM are less sensitive to blurriness, distortion, and temporal flickering since they measure local similarity. LPIPS and FvVDP are more suitable in our case, with one measuring the whole image perceptual similarity and the other measuring the video perceptual quality. We encourage readers to combine quantitative and image/video qualitative comparisons for better understanding.

5.3 Comparison against Baselines

To demonstrate the effectiveness of our method, we compare our method with state-of-the-art baselines under three different settings. Note that frame interpolation and G-buffer dependent extrapolation methods are under easier settings, which means they do not need to handle either disocclusions or motion estimation while our method needs to handle both. Even though our method still achieves comparable or better results than baselines in general.

UPR-Net [Jin et al. 2023] and IFR-Net [Kong et al. 2022] are state-of-the-art video interpolation methods that use optical flow-like methods to generate intermediate frames. Offline video interpolation methods [Reda et al. 2022; Zhang et al. 2023; Zhou et al. 2023] take more than 100 ms per frame, which is too slow to be used in real-time rendering and irrelevant to our task. DLSS 3.0 [NVIDIA 2022] is a commercial framework where the codes and details of implementation are unavailable, and it is difficult to obtain the intermediate results for comparison.

ExtraSS [Wu et al. 2023a] is a joint framework for super resolution and frame extrapolation in real-time rendering with G-buffers for the corresponding extrapolated frames. We use ExtraSS-E modules for comparison, which is the extrapolation part of ExtraSS. We choose ExtraSS-E as our baseline instead of LMV [Wu et al. 2023b] because the latter one requires even more additional G-buffers for rendered and extrapolated frames, which is even far away from our goal of G-buffer free frame extrapolation.

DMVFN [Hu et al. 2023] is a video future prediction method that uses current and previous frames to predict the future frame and can be considered as a G-buffer free frame extrapolation baseline.

UPR-Net, IFR-Net, and DMVFN are trained on a large-scale video dataset and we fine-tuned their pre-trained models on our datasets with a learning rate of 10^{-4} for 50 epochs. ExtraSS-E is trained on our datasets from scratch with the same settings as ours.

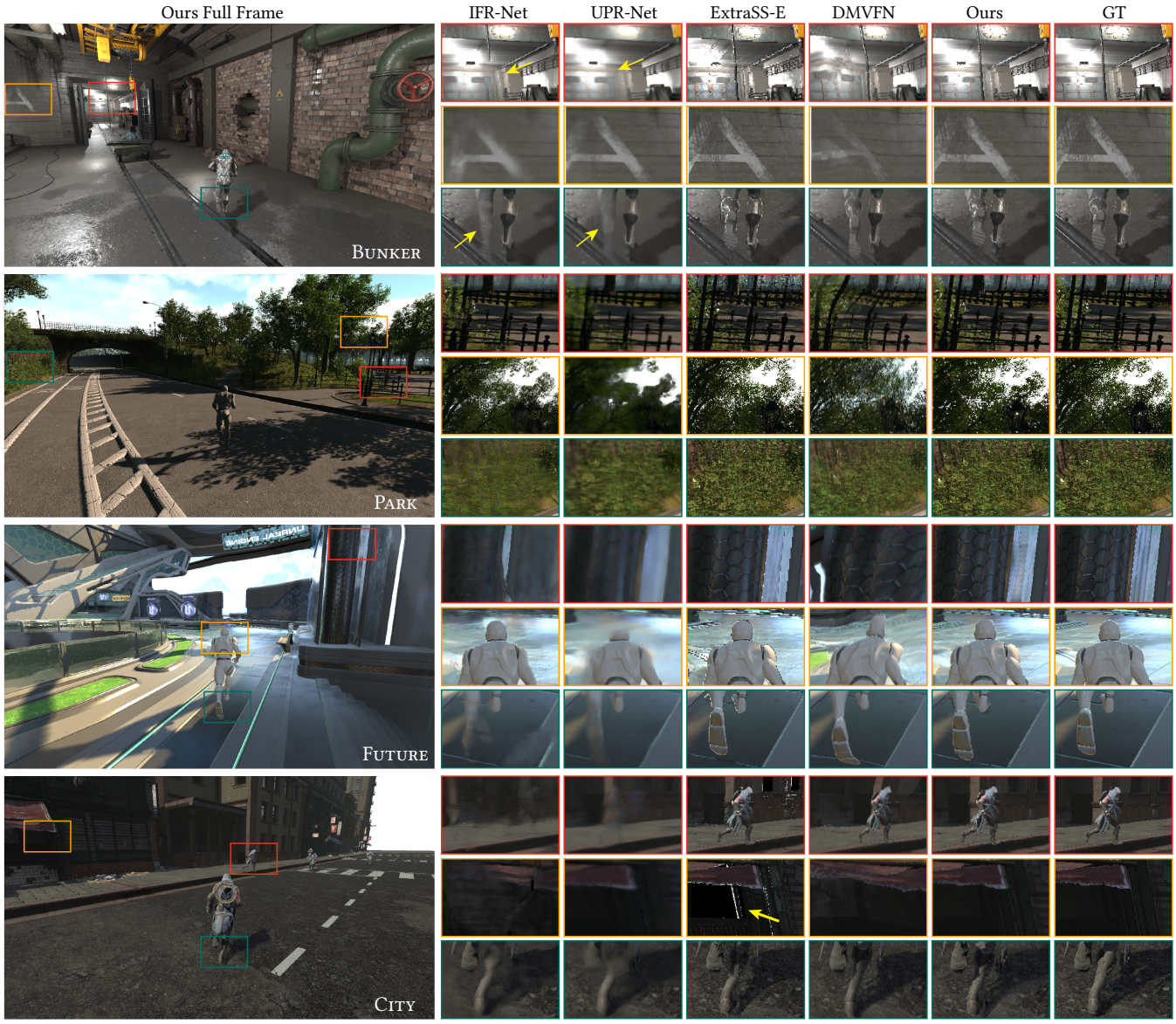

Fig. 8. Qualitative comparison in **trained** scenes between our method and baseline methods including DMVFN [Hu et al. 2023], UPR-Net [Jin et al. 2023], IFR-Net [Kong et al. 2022] and ExtraSS-E [Wu et al. 2023a]. DMVFN generates distorted results and cannot generate correct results if the information is missing from two given images. UPR-Net generates over-blurred results and misses thin geometries. ExtraSS-E generates overall good results but fails with transparent materials (windows in the second row). Our method generates detailed extrapolated frames closer to the ground truth with correct geometries and shadings.

5.3.1 Qualitative comparison. The qualitative comparison is shown in Fig. 8 (trained scenes) and Fig. 9 (not trained scenes). DMVFN generates highly distorted results when motion is large and can not generate correct results for the areas that do not have corresponding information in the prior two frames. Frame interpolation methods UPR-Net and IFR-Net cannot track the motion of thin geometries, so thin geometries are usually missing in this case. Besides, their estimated optical flows are not accurate enough to generate sharp results so their results are usually over-blurred or even severely distorted in some cases. ExtraSS-E uses ground truth G-buffers to guide the generation of extrapolated frames, which is usually more stable and contains more details. However, it fails with transparent materials (windows in City) and generates flickering results. Our method generates more stable frames with sharper details and less distortions.

5.3.2 Quantitative comparison. Table 3 shows the quantitative comparison against baselines. As discussed in Sec. 5.2, PSNR and SSIM metrics are not always reliable for evaluating the quality of the

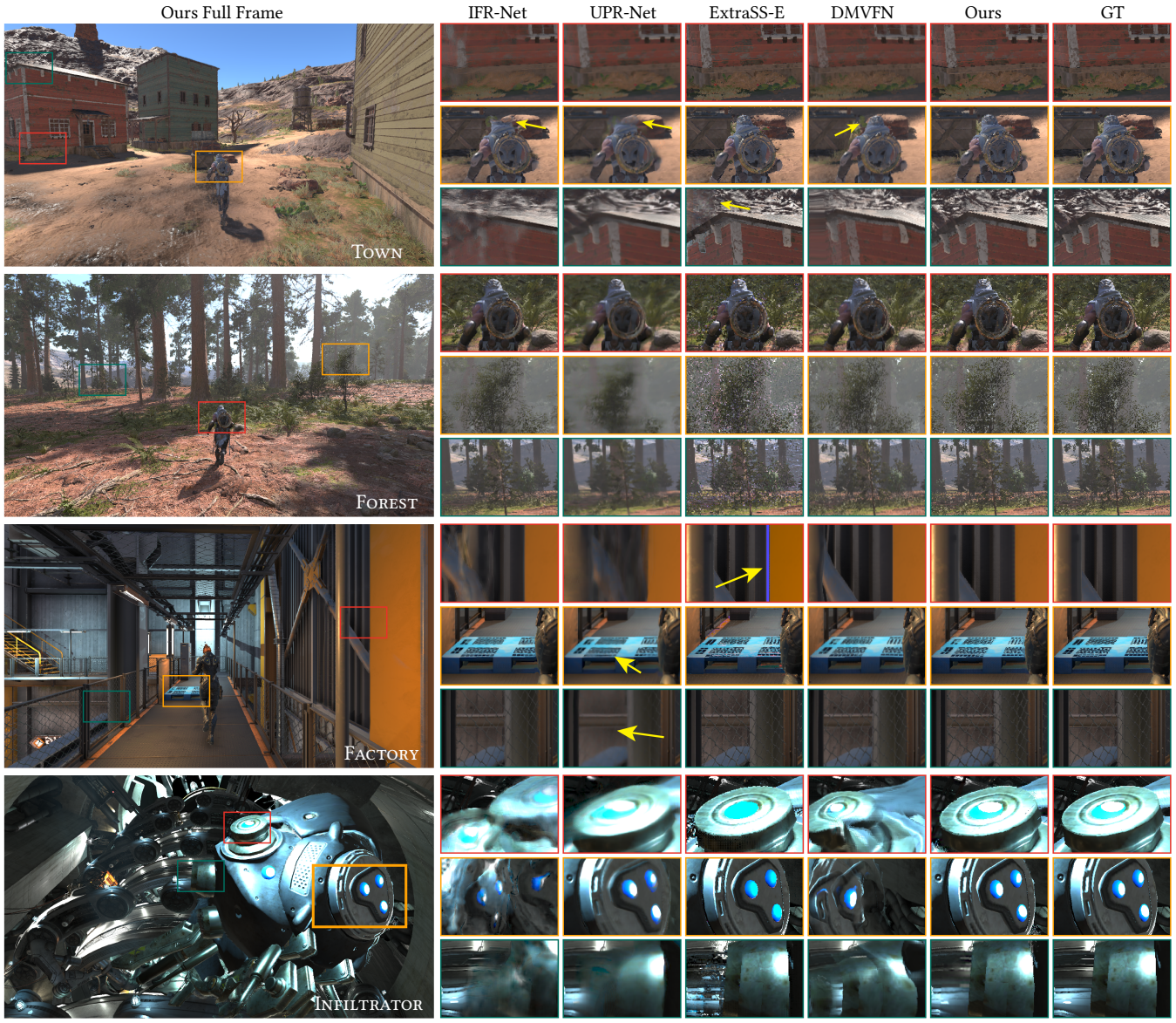

Fig. 9. Qualitative comparison in **test** scenes between our method and baseline methods including DMVFN [Hu et al. 2023], UPR-Net [Jin et al. 2023], IFR-Net [Kong et al. 2022] and ExtraSS-E [Wu et al. 2023a]. Our method still shows comparable or better visual quality with less distortion, blurriness, and artifacts, even though there are some big gaps between the training and test scenes.

generated frames. UPR-Net and IFR-Net show severe distortion and missing geometries as shown in previous qualitative comparisons, although they have higher PSNR and SSIM metrics in some scenes. ExtraSS-E shows marginally better results in scenes with complex geometries such as Forest and Park since G-buffers provide strong clues for the generation of extrapolated frames but the time of generating G-buffer in those scenes is usually long. Besides, it fails with scenes with translucent/transparent materials like City and Future. DMVFN shows significantly lower PSNR and SSIM metrics than other methods since it struggles in disocclusion areas and generates severely distorted results.

Besides PSNR and SSIM metrics, our method shows better results in LPIPS and FvVDP metrics, which are more perceptually suitable for evaluating the quality of the generated frames and more consistent with qualitative image and video comparison. Our method generates more stable and plausible frames than other baselines, which is essential in real-time rendering applications since people notice flickering and distortion more than pixel-level differences.

Based on our quantitative and qualitative evaluation, GFFE is significantly better than G-buffer free extrapolation baselines in all aspects, and shows comparable results with G-buffer dependent and

Table 3. Quantitative comparison with UPR-Net [Jin et al. 2023], IFR-Net [Kong et al. 2022], DMVFN [Hu et al. 2023] and ExtraSS-E [Wu et al. 2023a] under PSNR, SSIM, LPIPS, and FvVDP. Our method shows comparable quality with interpolation methods UPR-Net and IFR-Net, and G-buffer dependent extrapolation method ExtraSS-E under PSNR and SSIM metrics. Besides, our method shows better perceptual quality than interpolation methods and the G-buffer dependent method since these baselines are over-blurred, distorted, or flickering as analyzed in qualitative comparison. Our method also outperforms G-buffer free extrapolation baseline DMVFN in all aspects. The values of SSIM and LPIPS are scaled by 10^2. For PSNR, SSIM, and FvVDP, higher is better. For LPIPS, lower is better.

			Trained				Not trained				
	Type	Method	Bunker	Park	Future	City	Town	Forest	Factory	Infiltrator	Average
PSNR ↑	Interp.	IFR	25.55	18.37	28.66	26.16	27.41	19.62	22.40	25.78	24.24
		UPR	26.35	18.52	**28.75**	27.99	**27.75**	19.60	23.87	**26.12**	**24.87**
	G-buf Extrap.	ExSS-E	26.44	**24.75**	25.77	24.38	27.45	**21.71**	23.10	23.53	24.64
	Extrap.	DMVFN	23.79	16.89	23.69	24.77	24.44	17.00	20.38	22.84	21.73
		Ours	**27.84**	17.04	26.33	**29.77**	26.09	18.21	**24.50**	23.78	24.20
SSIM ↑	Interp.	IFR	86.78	71.43	**94.82**	84.84	91.16	70.23	84.66	91.21	84.39
		UPR	88.21	72.44	94.65	88.26	90.58	67.55	86.16	**91.77**	84.95
	G-buf Extrap.	ExSS-E	91.17	**86.79**	91.68	88.92	**92.32**	**78.57**	88.45	85.51	**87.93**
	Extrap.	DMVFN	82.39	61.18	87.46	81.43	81.63	51.97	76.25	86.39	76.09
		Ours	**93.50**	73.80	93.46	**93.49**	89.04	65.25	**89.55**	89.37	85.93
LPIPS ↓	Interp.	IFR	15.62	24.19	10.67	22.86	11.89	28.00	17.95	12.88	18.01
		UPR	22.49	42.44	17.15	24.05	22.99	52.81	26.15	19.19	28.41
	G-buf Extrap.	ExSS-E	12.22	14.63	14.73	17.38	**7.79**	17.78	15.72	24.26	15.56
	Extrap.	DMVFN	16.89	28.12	12.83	20.06	16.65	35.44	20.79	15.42	20.78
		Ours	**6.68**	**14.02**	**5.74**	**7.24**	7.98	**16.98**	**9.88**	**8.81**	**9.67**
FvVDP ↑	Interp.	IFR	7.98	6.77	5.36	7.20	8.35	7.07	6.84	7.77	7.17
		UPR	8.52	6.85	**5.40**	8.45	**8.75**	7.16	7.24	8.05	7.55
	G-buf Extrap.	ExSS-E	8.21	**7.40**	5.30	5.96	8.11	**7.38**	7.39	7.78	7.19
	Extrap.	DMVFN	7.25	6.57	5.37	6.85	7.88	6.43	6.49	7.56	6.80
		Ours	**8.65**	6.97	5.37	**8.58**	8.65	7.05	**7.87**	**8.09**	**7.65**

Table 4. Run time (ms) for all methods to generate 1080p frames, + means not including the time of generating G-buffers, see Table 6.

UPRNet	IFRNet	DMVFN	ExSS-E	Ours-Full
43.04	19.50	20.57	4.18+	**6.62**

Table 5. Run time (ms) breakdown for our framework under different resolutions. Misc mainly includes adjusting the display window and maintaining correct motion vectors.

	540p	720p	1080p
BG Collection	0.34	0.54	1.13
History Track	0.27	0.49	1.04
Misc	0.09	0.14	0.37
BG Projection	0.16	0.32	0.76
Position Pred.	0.07	0.11	0.22
Warp	0.51	0.76	0.80
SCN	0.90	1.30	2.30
Total	2.34	3.66	6.62

interpolation methods with better perceptual quality in LPIPS and FvVDP metrics.

5.4 Generalization

Our framework is trained on four scenes and tested on eight scenes, where four scenes are never shown during training. Despite the limited dataset, our method generates stable and plausible results in all scenes. This is because our hybrid modules are robust to different scenes and can handle disocclusions and motion estimation well instead of using a single neural network to handle all problems, which requires a large-scale dataset for training. Therefore, we consider our method to have good generalization ability and robustness to various different scenes.

5.5 Performance

Performance is an important factor for real-time rendering applications. We used a machine with NVIDIA RTX 4070Ti Super GPU and Ryzen 9 5900X CPU for inference. The non-neural modules (GAE) of our method are implemented in NVIDIA Falcor [Kallweit et al. 2022]

Table 6. G-buffer generation time (ms) under 1080p for different scenes. For non-experimented scenes in products, the time may even exceed 10 ms.

	Bunker	Park	Future	City
Time	0.35	8.23	2.85	0.40
	Town	Forest	Factory	Infiltrator
Time	2.02	2.61	1.91	0.96

Table 7. Ablation study on our designed modules. Numbers are averaged for all scenes. SSIM and LPIPS numbers are scaled by 10^2. ME = Motion Estimation, BGC = Hierarchical Background Collection, AW = Adaptive Render Window, SCN = Shading Correction Network, FM = Focus Mask.

	PSNR↑	SSIM↑	LPIPS↓	FvVDP↑
w/o ME	23.44	85.21	10.36	7.57
w/o BGC	24.11	85.91	9.79	7.62
w/o AW	24.15	85.91	9.71	7.62
w/o SCN	23.93	85.86	**9.23**	7.60
w/o FM	23.86	81.39	35.63	7.49
Ours Full	**24.20**	**85.93**	9.67	**7.65**

renderer. All neural networks, including baselines, are trained under the PyTorch framework and converted into TensorRT [NVIDIA 2021] with FP16 precision for inference.

Table 5 shows the breakdown run time of our method under different resolutions. Note that our method is designed to be applied in the post-processing stage and the performance is not affected by the scene's complexity.

Table 4 shows the run time for all methods. Previous frame interpolation methods UPR-Net and IFR-Net and extrapolation method DMVFN are much slower than ours since neural networks are usually slow compared to heuristic methods. ExtraSS-E is faster than our method since it uses G-buffers to guide the generation of extrapolated frames. However, the time of generating G-buffers is not included in the run time of ExtraSS-E, which varies depending on the scene's complexity. Table 6 shows the time of generating G-buffers for different scenes, where complex geometry scenes like Forest and Park take a longer time to generate G-buffers. Note that the scenes in which ExtraSS-E is better than ours are usually scenes with complex geometries.

Although some dedicated hardware or software optimizations could accelerate the run time performance, all methods are tested under the same environment and settings without dedicated optimizations, so any optimizations applied to baselines can also be applied to our method to achieve better performance.

6 Ablation Study

Our framework is a complete pipeline that consists of several modules for G-buffer free frame extrapolation. In this section, we analyze the effectiveness and importance of each module in our framework to demonstrate the necessity of each module. Table 7 shows the quantitative metrics of removing our designed modules, and more qualitative results are included in the following sections and the supplementary video.

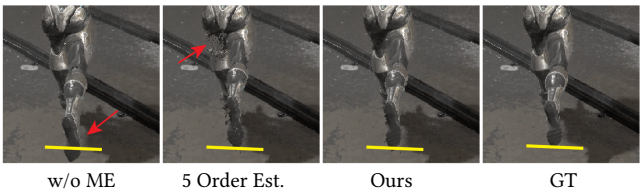

Fig. 10. Ablation study of the motion estimation (ME) module. Without this module, the geometries do not move. With higher-order polynomials to estimate the motion, it diverges and is unstable. Our approach estimates the motion of dynamic objects more plausibly.

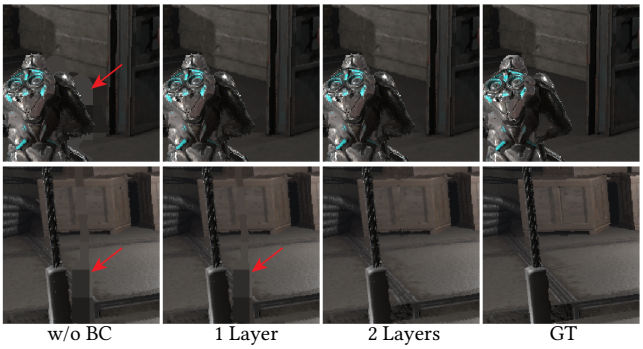

Fig. 11. Ablation study of the layered background collection module. Without background collection, the disocclusion areas are not handled at all. With one layer background, it only captures the background behind dynamic objects. With our two-layers background collection, it not only recovers disocclusions behind dynamic objects, but also static disocclusions behind static objects.

6.1 Motion Estimation

Motion estimation tracks the motion of dynamic fragments and projects fragments to extrapolated frames. Linear motion in world space estimates the next world position in the extrapolated frames. Higher-order polynomials could be used but with worse quality in our experiments. As shown in Fig. 10, dynamic objects are not moving without the motion estimation module. Higher-order polynomial estimation diverges and generates artifacts. Our module efficiently generates plausible motions for dynamic objects.

6.2 Background Collection

Background collection addresses static disocclusions and dynamic disocclusions. The results are usually of lower quality without such modules and directly guessing what is in the disocclusion areas. Our layered background collection module collects multiple layers background to handle different levels disocclusions. Fig. 11 shows the comparison between results with and without background collection. Our complete background collection module fixes not only the disocclusions behind dynamic objects but also the disocclusions behind the static objects due to camera motion.

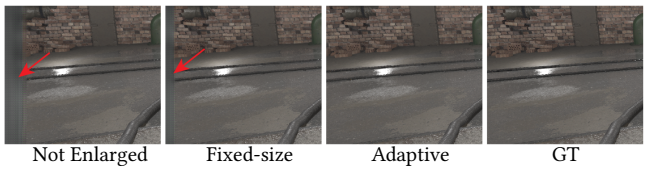

Fig. 12. Ablation study of adaptive rendering windows. Invalid regions appear at the boundary of the image without the adaptive windows. Fixed-size enlarged rendering window contains more redundant and less useful information for extrapolation. Our adaptive strategy can adjust the rendering window dynamically for better extrapolation.

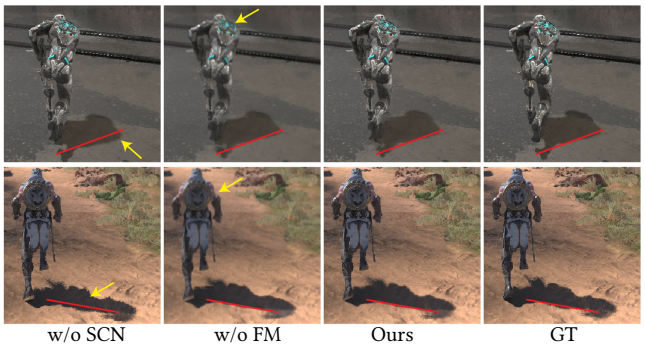

Fig. 13. Ablation study of the shading correction network (SCN). FM = Focus Mask. Without SCN, non-geometric motions are not tracked, so shadows are not moving in the extrapolated frame. Without the focus mask, directly predicting the final refined frame blurs the whole image. Our SCN with focus mask not only fixes non-geometric motion but also keeps sharp details.

6.3 Adaptive Rendering Window

We compare our adaptive strategy with fixed enlarged windows and not enlarged windows. Fig. 12 shows the comparison between these three methods. Our method covers more disocclusions since our rendering windows are adaptively adjusted based on camera motion.

6.4 Shading Correction Network

Our SCN module mainly fixes the lagging issue of non-geometries motion including shadows and reflections. As discussed in previous work [Guo et al. 2021], although such effects have a small impact on metrics or even slightly worse (LPIPS), it is noticeable in human perception and important for high quality rendering.

Fig. 13 shows the comparison between not using SCN, without the focus mask, and with our full SCN module. Without SCN, the shadows and reflections are not moving due to missing motion, leading to a lower frame rate experience in those areas. Without the focus mask, the neural network tries to refine the whole image, which blurs the overall details. Our full SCN module can detect the areas that need to be refined and only refine those areas without blurring other areas. For better visualization and comparison of this ablation study, please refer to the supplementary video to see how it affects the final results for continuous frames.

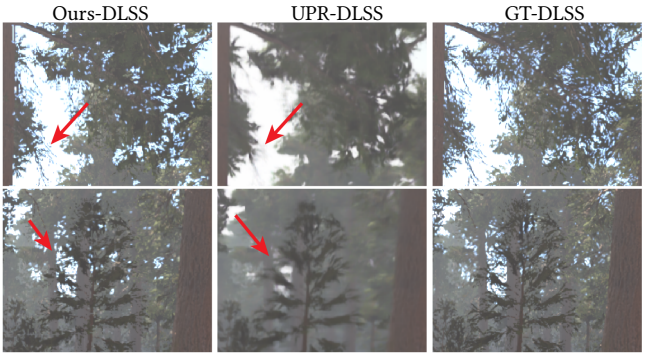

Fig. 14. Results of integrating DLSS 2 with ours and UPR [Jin et al. 2023]. Our results contain more details while UPR with DLSS 2 still generates blurred results.

7 Discussion

7.1 Anti-aliasing and Super Resolution

Our framework, unlike UPR-Net, IFR-Net, and DMVFN, generates not only extrapolated shaded frames, but also the corresponding depth buffer and motion vectors between the extrapolated frames and rendered frames. This indicates that the generated frames can be considered the same as other rendered frames to apply additional anti-aliasing or super resolution techniques.

Super sampling techniques, including DLSS [Liu 2020], XeSS [Intel 2022], and FSR [AMD 2021], have shown high quality results in generating higher resolution frames from lower resolution frames efficiently, which are widely used in real-time rendering to improve the visual quality. Our method with generated depth and motion vectors can be easily integrated with such super resolution techniques to generate higher quality frames. Fig. 14 shows the comparison between our method and UPR [Jin et al. 2023] of using DLSS on Forest with complex geometries. Our results contain more details while UPR tends to over-blur them. Ground truth depth and motion vectors are used for baselines in to use DLSS.

7.2 Practical Choices

We show breakdown performance and ablation studies in the previous section to demonstrate the usage of each module. Each module in our framework is relatively independent and can be removed or replaced with better modules in the future if needed. For example, for low end devices such as mobiles, the neural network module SCN could be removed since the shading changes are usually simpler, so the integration is easier and performance is better with some degradation in quality. Our framework is flexible and can be adjusted in various applications based on needs.

7.3 Limitations

As noted throughout the paper, our G-buffer free extrapolation method has much fewer inputs compared to G-buffer dependent extrapolation (missing G-buffers) and interpolation (missing future frames). Therefore, although with comparable quality overall, our method still has limitations. We analyze them below and show corresponding artifacts in Fig. 15.

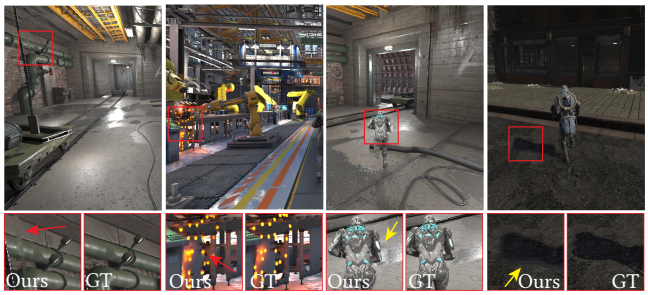

Fig. 15. Failure cases of our framework include uncollected disocclusions, effects without depth, shading changes in disocclusions, and imperfect shading correction.

Uncollected disocclusions. Our background collection module tries to find information from previous frames to fill in the disocclusions. However, it fails when the disocclusion areas have never been shown before and are not the out-of-screen areas (Fig. 15 the first column).

Effects without depth. Our framework relies on depth to calculate correct motions and projections. Some effects, including UI and particles, do not have such information, so our framework does not attempt to calculate correct positions in extrapolated frames (Fig. 15 the second column). One possible solution could be separating these effects into other passes and combining them with our extrapolated frames.

Shading changes in disocclusions. As shown in the third column of Fig. 15, the shading of background collected fragments can be incorrect due to view direction changes, dynamic lighting, etc. We currently do not specifically train our shading correction network to deal with this and leave it for future work.

Imperfect shading correction. Since our method lacks information from G-buffers and future frames compared to the other two types of methods, estimating refined shadings such as shadows is more complicated. As a result, the outcomes of such refined shadings are sometimes blurred (Fig. 15, the fourth column). A better shading correction module is left for future work.

8 Conclusion

We have presented a G-buffer free extrapolation method, GFFE, for low-latency real-time rendering. We addressed three challenges of G-buffer free extrapolation tasks by our designed modules: motion estimation, background collection, adaptive rendering windows, and shading correction network.

We evaluate GFFE on diverse scenes and show high quality extrapolation results that demonstrate robustness and generality. The proposed modules provide efficient frame generation without additional latency and extra G-buffers in the real-time rendering context. Our framework outperforms G-buffer free extrapolation baselines and is comparable with other frame generation methods, including frame interpolation and G-buffer dependent frame extrapolation, with better performance.

In the future, apart from improving the limitations above, GFFE may be worth exploring in the context of VR/AR and streaming applications. It can also be extended to perform multiple frame extrapolation by passing an extrapolation factor α to the shading correction network to refine the shading motion in different magnitudes to boost the performance further.

Acknowledgments

We thank the anonymous reviewers for their valuable suggestions. This project is solely sponsored by Intel, and Ling-Qi Yan is also supported by gift funds from Adobe, Linctex, Meta, and XVerse.

References

AMD. 2021. AMD FidelityFX™ Super Resolution. https://www.amd.com/en/technologies/fidelityfx-super-resolution Accessed: 2023-05-23.

AMD. 2022. AMD FidelityFX™ Super Resolution 3. https://gpuopen.com/fidelityfx-super-resolution-3/ https://gpuopen.com/fidelityfx-super-resolution-3/ [Accessed: 2024-01-24].

Dmitry Andreev. 2010. Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for" free". In *ACM SIGGRAPH Talks*. 1–1.

Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. 2019. Depth-aware video frame interpolation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3703–3712.

Huw Bowles, Kenny Mitchell, Robert W Sumner, Jeremy Moore, and Markus Gross. 2012. Iterative image warping. In *Computer graphics forum*, Vol. 31. Wiley Online Library, 237–246.

Karlis Martins Briedis, Abdelaziz Djelouah, Mark Meyer, Ian McGonigal, Markus Gross, and Christopher Schroers. 2021. Neural frame interpolation for rendered content. *ACM Trans. Graph.* 40, 6 (2021), 1–13.

Karlis Martins Briedis, Abdelaziz Djelouah, Raphaël Ortiz, Mark Meyer, Markus Gross, and Christopher Schroers. 2023. Kernel-Based Frame Interpolation for Spatio-Temporally Adaptive Rendering. In *ACM Trans. Graph. (SIGGRAPH)*. 1–11.

Pierre Charbonnier, Laure Blanc-Feraud, Gilles Aubert, and Michel Barlaud. 1994. Two deterministic half-quadratic regularization algorithms for computed imaging. In *Proc. international conference on image processing*, Vol. 2. 168–172.

Piotr Didyk, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel. 2010. Perceptually-motivated real-time temporal upsampling of 3D content for high-refresh-rate displays. In *Comp. Graph. Forum*, Vol. 29. 713–722.

Epic Games. 2022. *Unreal Engine.* https://www.unrealengine.com

Jie Guo, Xihao Fu, Liqiang Lin, Hengjun Ma, Yanwen Guo, Shiqiu Liu, and Ling-Qi Yan. 2021. ExtraNet: Real-Time Extrapolated Rendering for Low-Latency Temporal Supersampling. *ACM Trans. Graph.*, Article 278 (2021).

Yu-Xiao Guo, Guojun Chen, Yue Dong, and Xin Tong. 2022. Classifier Guided Temporal Supersampling for Real-time Rendering. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 237–246.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1026–1034. https://doi.org/10.1109/ICCV.2015.123

Xiaotao Hu, Zhewei Huang, Ailin Huang, Jun Xu, and Shuchang Zhou. 2023. A dynamic multi-scale voxel flow network for video prediction. In *Proc. IEEE CVPR*. 6121–6131.

Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. 2022. Real-time intermediate flow estimation for video frame interpolation. In *European Conference on Comp. Vision*. 624–642.

Intel. 2022. Intel® Arc™- Xe Super Sampling. https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/arc/technology/xess.html Accessed: 2023-05-23.

Xin Jin, Longhai Wu, Jie Chen, Youxin Chen, Jayoon Koo, and Cheul-hee Hahm. 2023. A Unified Pyramid Recurrent Network for Video Frame Interpolation. In *Proc. IEEE CVPR*. 1578–1587.

Nima Khademi Kalantari, Ravi Ramamoorthi, et al. 2017. Deep high dynamic range imaging of dynamic scenes. *ACM Trans. Graph.* 36, 4 (2017), 144–1.

Simon Kallweit, Petrik Clarberg, Craig Kolb, Tom'aš Davidovič, Kai-Hwa Yao, Theresa Foley, Yong He, Lifan Wu, Lucy Chen, Tomas Akenine-Möller, Chris Wyman, Cyril Crassin, and Nir Benty. 2022. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor https://github.com/NVIDIAGameWorks/Falcor.

Joohwan Kim, Pyarelal Knowles, Josef Spjut, Ben Boudaoud, and Morgan Mcguire. 2020. Post-render warp with late input sampling improves aiming under high latency conditions. *Proc. ACM Comp. Graph. and Interactive Techniques* 3, 2 (2020), 1–18.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Lingtong Kong, Boyuan Jiang, Donghao Luo, Wenqing Chu, Xiaoming Huang, Ying Tai, Chengjie Wang, and Jie Yang. 2022. Ifrnet: Intermediate feature refine network for efficient frame interpolation. In *Proc. IEEE CVPR*. 1969–1978.

Sungkil Lee, Younguk Kim, and Elmar Eisemann. 2018. Iterative Depth Warping. *ACM Trans. Graph.* 37, 5, Article 177 (oct 2018), 13 pages. https://doi.org/10.1145/3190859
Zhan Li, Carl S Marshall, Deepak S Vembar, and Feng Liu. 2022. Future Frame Synthesis for Fast Monte Carlo Rendering. In *Graph. Interface*.
Edward Liu. 2020. DLSS 2.0-Image reconstruction for real-time rendering with deep learning. In *Nvidia GPU Tech. Conf. (GTC)*.
Rafał K. Mantiuk, Gyorgy Denes, Alexandre Chapiro, Anton Kaplanyan, Gizem Rufo, Romain Bachy, Trisha Lian, and Anjul Patney. 2021. FovVideoVDP: a visible difference predictor for wide field-of-view video. *ACM Trans. Graph.*, Article 49 (jul 2021), 19 pages. https://doi.org/10.1145/3450626.3459831
William R Mark, Leonard McMillan, and Gary Bishop. 1997. Post-rendering 3D warping. In *Proc. Symp. Interactive 3D Graphics*. 7–ff.
Simon Meister, Junhwa Hur, and Stefan Roth. 2018. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *Proc. AAAI conference on artificial intelligence*, Vol. 32.
NVIDIA. 2021. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt
NVIDIA. 2022. NVIDIA DLSS 3: AI-Powered Performance Multiplier Boosts Frame Rates By Up To 4X. https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/ https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/ [Accessed: 2024-01-24].
Oculus. 2016. Asynchronous Spacewarp. https://developer.oculus.com/blog/asynchronous-spacewarp/
Fitsum Reda, Janne Kontkanen, Eric Tabellion, Deqing Sun, Caroline Pantofaru, and Brian Curless. 2022. Film: Frame interpolation for large motion. In *European Conference on Computer Vision*. Springer, 250–266.
Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and Hans-Peter Seidel. 2016. Proxy-guided image-based rendering for mobile devices. In *Comp. Graph. Forum*, Vol. 35. 353–362.
Andre Schollmeyer, Simon Schneegans, Stephan Beck, Anthony Steed, and Bernd Froehlich. 2017. Efficient hybrid image warping for high frame-rate stereoscopic rendering. *IEEE Trans. Vis. and Comp. Graph.* 23, 4 (2017), 1332–1341.
Josef Spjut, Ben Boudaoud, Kamran Binaee, Jonghyun Kim, Alexander Majercik, Morgan McGuire, David Luebke, and Joohwan Kim. 2019. Latency of 30 ms Benefits First Person Targeting Tasks More Than Refresh Rate Above 60 Hz. In *SIGGRAPH Asia 2019 Technical Briefs (SA '19)*. Association for Computing Machinery, New York, NY, USA, 110–113. https://doi.org/10.1145/3355088.3365170
Josef B. Spjut, Ben Boudaoud, and Joohwan Kim. 2021. A Case Study of First Person Aiming at Low Latency for Esports. *CoRR* abs/2105.10498 (2021). arXiv:2105.10498 https://arxiv.org/abs/2105.10498
J. M. P. van Waveren. 2016. The asynchronous time warp for virtual reality on consumer hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology* (Munich, Germany) *(VRST '16)*. Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/2993369.2993375
Songyin Wu, Sungye Kim, Zheng Zeng, Deepak Vembar, Sangeeta Jha, Anton Kaplanyan, and Ling-Qi Yan. 2023a. ExtraSS: A Framework for Joint Spatial Super Sampling and Frame Extrapolation. In *ACM Trans. Graph. (SIGGRAPH Asia)*. Article 92.
Zhizhen Wu, Chenyu Zuo, Yuchi Huo, Yazhen Yuan, Yifan Peng, Guiyang Pu, Rui Wang, and Hujun Bao. 2023b. Adaptive Recurrent Frame Prediction with Learnable Motion Vectors. In *ACM Trans. Graph. (SIGGRAPH Asia)*. Article 10.
Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural supersampling for real-time rendering. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 142–1.
Lei Yang, Yu-Chiu Tse, Pedro V Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L Wilkins. 2011. Image-based bidirectional scene reprojection. In *ACM Trans. Graph. (SIGGRAPH Asia)*. 1–10.
Sipeng Yang, Qingchuan Zhu, Junhao Zhuge, Qiang Qiu, Chen Li, Yuzhong Yan, Huihui Xu, Ling-Qi Yan, and Xiaogang Jin. 2024. Mob-FGSR: Frame Generation and Super Resolution for Mobile Real-Time Rendering. In *ACM SIGGRAPH 2024 Conference Papers (SIGGRAPH '24)*. Association for Computing Machinery, Article 64, 11 pages. https://doi.org/10.1145/3641519.3657424
Zheng Zeng, Shiqiu Liu, Jinglei Yang, Lu Wang, and Ling-Qi Yan. 2021. Temporally Reliable Motion Vectors for Real-time Ray Tracing. In *Comp. Graph. Forum*, Vol. 40. 79–90.
Guozhen Zhang, Yuhan Zhu, Haonan Wang, Youxin Chen, Gangshan Wu, and Limin Wang. 2023. Extracting motion and appearance via inter-frame attention for efficient video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5682–5692.
Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proc. IEEE CVPR*.
Kun Zhou, Wenbo Li, Xiaoguang Han, and Jiangbo Lu. 2023. Exploring motion ambiguity and alignment for high-quality video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 22169–22179.

A Limitation of using G-buffers in extrapolated frames

As discussed in Sec. 3.1, G-buffers are not available or become the bottleneck under the following cases:

- **Availability:** Some types of G-buffers used in previous extrapolation methods, including albedo, roughness, and metallic, are only available for deferred rendering pipelines. Forward rendering pipelines, which are widely used in smartphones, consoles, and even personal computer platforms, do not provide such G-buffers.
- **Complexity:** G-buffer generation is the bottleneck in some real-time applications. Simulation heavy games require complex simulation processes so generating G-buffers is quite time consuming. Besides, some modern games generate high quality G-buffers with low quality shading and then modulate the shading with the G-buffers to render final detailed images, where the generation of G-buffers consumes the majority of the time.
- **Memory requirements:** Even if the generation of G-buffers is not the bottleneck, it still requires additional memory to store them with additional cost in multiple aspects.

In these cases, the G-buffer dependent methods [Guo et al. 2021; Wu et al. 2023a,b] are limited.

B History tracking algorithm

Here are the details of the history collection algorithm. M^{dyn} is the dynamic mask where the value of dynamic pixels is one.

ALGORITHM 1: History tracking with static test

Data: Pixel position x, Current frame depth $\{D_t\}$, Current motion vector $V_{t\to t-1}$, Current and previous camera poses $\{C_t, C_{t-1}\}$, previous history trajectory P', length of history trajectory k
Result: History trajectory P, Dynamic Mask M_t^{dyn}

$p \leftarrow \text{unproject}(x, D_t, C_t)$; // cur world position
$\hat{x} \leftarrow \text{project}(p, C_{t-1})$; // previous position
$x' = x + V_{t\to t-1}[x]$; // previous position
if $\|\hat{x} - x'\|_2 > \varepsilon$ **then** // static test
  for *i = 1* **to** *k-1* **do**
    $P_i[x] = P'_{i-1}[x']$;
  end
  $P_0[x] = p$;
  $M_t^{\text{dyn}}[x] \leftarrow 1$;
end
else // fragment is static
  for *i = 0* **to** *k-1* **do**
    $P_i[x] = p$;
  end
  $M_t^{\text{dyn}}[x] \leftarrow 0$;
end

C Adaptive rendering window

After obtaining current camera pose C_t and estimated next camera pose $\bar{C}_{t+\alpha}$, a virtual plane will be put in front of the current

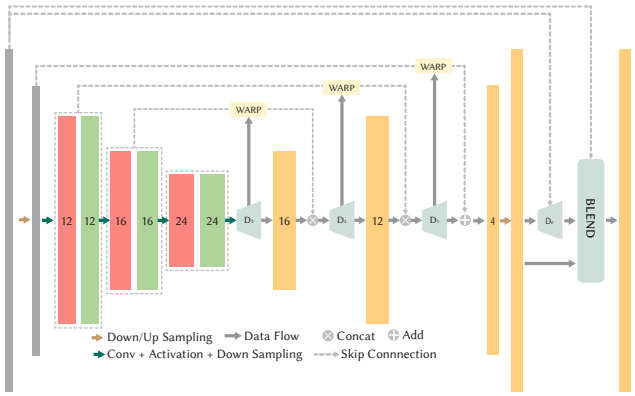

Fig. 16. The network structure of shading correction network. The input is down-sampled at first to improve performance.

camera along the `lookat` direction with distance d to assist calculating approximated viewport in the extrapolated frames. By calculating the intersections of four corners of camera C_t's view frustum, we get the coordinates of four intersections and their corresponding 2D axis-aligned bounding box on the plane, denoted as $r = (x_{\min}, y_{\min}, x_{\max}, y_{\max})$. Similarly, we calculate the axis-aligned bounding box of estimated camera $\tilde{C}_{t+\alpha}$ on the same virtual plane, denoting as $\bar{r} = (\bar{x}_{\min}, \bar{y}_{\min}, \bar{x}_{\max}, \bar{y}_{\max})$. Then, we can calculate the enlarged size of rendering windows based on relative sizes of r and $\bar{r}$. Assume the original rendering window is the rectangle $(-1, -1, 1, 1)$, the adaptive window of current frame (u_0, v_0, u_1, v_1) is calculated by:

$$\begin{cases} u_0 = \min(-1, -\bar{x}_{\min}/x_{\min}) \\ v_0 = \min(-1, -\bar{y}_{\min}/y_{\min}) \\ u_1 = \max(1, \bar{x}_{\max}/x_{\max}) \\ v_1 = \max(1, \bar{y}_{\max}/y_{\max}) \end{cases} \tag{5}$$

Note that the virtual plane is put in front of the current camera, so the bounding box of the current camera on the virtual plane always satisfies $x_{\min} = -x_{\max} < 0$ and $y_{\min} = -y_{\max} < 0$.

D Shading Correction Network

Network structure. SCN is a flow-based network with gradually predicted flows to warp intermediate features. The structure of SCN is shown in Fig. 16. We use PReLU [He et al. 2015] as activation functions and each decoder module contains a sequence of 2D convolution, residual block, and 2D convolution with PReLU activation functions between them.

The output contains a predicted focus mask and a refined image, and the final output is the blending between the refined image and the input GAE image.

Loss functions. To train our SCN, we use the following loss functions to cover various aspects of the output.

Intermediate feature loss $\mathcal{L}_f$ [Kong et al. 2022] constrains the intermediate features to better align the non-geometric flows from coarse to fine levels. It is defined as:

$$\mathcal{L}_f = \sum_{k=1}^{3} \mathcal{L}_{cen}(\bar{\phi}_k, \phi_k) \tag{6}$$

where $\mathcal{L}_{\text{cen}}$ is the census loss [Meister et al. 2018] and $\bar{\phi}_k$ and ϕ_k are the intermediate features k of extrapolated frames and ground truth frames from the encoder.

Focus mask loss is the key part of our SCN module to predict a correct focus mask and maintain sharp results. It is defined as:

$$\mathcal{L}_{\text{focus}} = \left\| \bar{M}_{\text{focus}} - M_{\text{focus}} \right\|_2 \tag{7}$$

Reconstruction loss $\mathcal{L}_{\text{recon}}$ is calculated by Charbonnier loss [Charbonnier et al. 1994] between the final predicted image and the ground truth image. The VGG perceptual loss $\mathcal{L}_{\text{vgg}}$ is used to keep the details of extrapolated frames. The final loss function is formulated as

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda_f \mathcal{L}_f + \lambda_{\text{focus}} \mathcal{L}_{\text{focus}} + \lambda_{\text{vgg}} \mathcal{L}_{\text{vgg}} \tag{8}$$

where we set $\lambda_f = 0.01$, $\lambda_{\text{focus}} = 1.0$, $\lambda_{\text{vgg}} = 0.01$ in our experiments.

Data preparation. During training, we crop the original images into 256×256 patches to train the network. Since our GAE module provides almost correct geometries, the majority of areas in extrapolated frames are correct, which is less useful for training the network. Therefore, we first randomly crop 10^6 patches from the training dataset and then sort the crops based on the areas of focus mask M_{focus}. We keep the top 15% patches and randomly select the other 3% patches for training. We still evaluate on full resolution images during the inference process. All color images in the linear space will be first tone-mapped by μ-Law [Kalantari et al. 2017] tone-mapper before feeding into the network and the final output will be inverse tone-mapped to the linear space. All losses are calculated in the tone-mapped space.

Training. We train our model on the cropped dataset with batch size 256 for 300 epochs. We use Adam [Kingma and Ba 2014] optimizer with learning rate starting from 10^{-4} and gradually decay to 10^{-5} during the training. We use PyTorch to implement our network and train it on four NVIDIA A6000 GPUs.

E Discussion with Mob-FGSR

Concurrent work Mob-FGSR [Yang et al. 2024] shares some similar ideas with our method using a heuristic method to efficiently generate new frames. However, there are three main differences between our method and Mob-FGSR, which reflect three key challenges (motion estimation, disocclusions, and non-geometric motion tracking) as discussed in Sec. 3.2. Our world space motion tracking is more robust for motion prediction since it eliminates the effect of perspective projection. The hole filling algorithm in Mob-FGSR may fail when backgrounds are not repetitive textures while our method caches actual geometries. Our SCN model fixes the incorrect shading/shadow in extrapolated frames, while Mob-FGSR does not handle this part. In general, our framework is more robust and general in various real-time rendering applications.