PatchEX: High-Quality Real-Time Temporal Supersampling through Patch-based Parallel Extrapolation

AKANKSHA DIXIT, Electrical Engineering, Indian Institute of Technology Delhi, India
SMRUTI R. SARANGI, Electrical Engineering, Indian Institute of Technology Delhi, India

High-refresh rate displays have become very popular in recent years due to the need for superior visual quality in gaming, professional displays and specialized applications such as medical imaging. However, high-refresh rate displays alone do not guarantee a superior visual experience; the GPU needs to render frames at a matching rate. Otherwise, we observe disconcerting visual artifacts such as screen tearing and stuttering. Real-time frame generation is an effective technique to increase frame rates by predicting new frames from other rendered frames. There are two methods in this space: interpolation and extrapolation. Interpolation-based methods provide good image quality at the cost of a higher runtime because they also require the next rendered frame. On the other hand, extrapolation methods are much faster at the cost of quality. This paper introduces *PatchEX*, a novel frame extrapolation method that aims to provide the quality of interpolation at the speed of extrapolation. It smartly segments each frame into foreground and background regions and employs a novel neural network to generate the final extrapolated frame. Additionally, a wavelet transform (WT)-based filter pruning technique is applied to compress the network, significantly reducing the runtime of the extrapolation process. Our results demonstrate that *PatchEX* achieves a 61.32% and 49.21% improvement in PSNR over the latest extrapolation methods ExtraNet and ExtraSS, respectively, while being 3× and 2.6× faster, respectively.

CCS Concepts: • **Computer systems organization → Real-time systems**; • **Computing methodologies → Rendering**; **Machine learning**.

Additional Key Words and Phrases: Extrapolation, Warping, G-buffer, Perceptual sensitivity

ACM Reference Format:
Akanksha Dixit and Smruti R. Sarangi. 2025. *PatchEX*: High-Quality Real-Time Temporal Supersampling through Patch-based Parallel Extrapolation. 1, 1 (September 2025), 19 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the last few years, there has been a significant growth in the demand for high-refresh rate displays. Refresh rates have reached 360 Hz for major monitor brands. This surge is driven by the need for enhanced visual quality in various market segments such as gaming, professional displays (used in fields like esports) and specialized applications such as medical imaging and scientific visualization [Gembler et al. 2018; Huhti 2019; LED 2025; Murakami et al. 2021]. The global gaming monitor market alone was valued at around USD 9.51 billion in 2022 and is projected to grow to approximately USD 16.04 billion by 2030 with a compound annual growth rate (CAGR) of 6.76% between 2023 and 2030 [gam 2023]. The reason for this trend is because low-refresh rate displays sometimes exhibit various visual artifacts such as judder (non-continuous motion perception) and motion blur during high-speed motion [Han et al. 2022]. High-frequency displays thus aim to deliver a smooth and seamless experience by eliminating these artifacts.

GPU is the bottleneck: It is crucial to acknowledge that having high-frequency displays alone may not always guarantee smooth performance unless the frame rendering rate matches the refresh rate. When the rendering rate is lower than the refresh rate, visual artifacts such as screen tearing and stuttering can occur [Denes et al. 2020]. Technologies like G-Sync address this by enabling variable refresh rates that synchronize with the rendering rate, though this may result in underutilization of the display's full refresh rate potential [Slavenburg et al. 2020]. Therefore, it is essential for the GPU to render frames at a matching rate, which is seldom feasible. As graphics engineers continue to incorporate increasingly complex effects into graphics applications to enhance realism, the rendering process becomes more intricate and time consuming (please refer to A.4). Several studies have shown the variation in the rendering rate and its impact on the quality of experience for users [Klein et al. 2024; Liu et al. 2023; Sabet et al. 2020; Xu and Claypool 2024]. This necessitates the exploration of strategies to produce additional frames post-rendering such that the rate of frame generation is equal to the refresh rate of the monitor.

Frame generation fills in frames missed by the GPU: One of the most impactful approaches to increase the frame rate is *frame generation*, which involves predicting frames using information from the next and previously rendered frames [Guo et al. 2021; He et al. 2024; Niklaus and Liu 2020; Wu et al. 2023a,b; Zhang et al. 2023]. The core concept here is that since rendering new frames is time-consuming, we can expedite the process by predicting new frames from previously rendered ones or the next frame (in temporal sequence) and interleave the frames at the display device. This boosts the frame rate and achieves rate matching. For frame generation to be effective, it is important to ensure that the prediction time is shorter than the rendering time and that the predicted frame is of acceptable quality. Particularly in real-time systems like virtual reality applications and games, minimizing runtime and ensuring good quality are of utmost importance.

Interpolation: high quality, high runtime ↔ Extrapolation: low quality, low runtime: In the field of frame generation, two primary methodologies exist: *interpolation* [Niklaus and Liu 2020; Wu et al. 2023b; Zhang et al. 2023] and *extrapolation* [Guo et al. 2021; He et al. 2024; Wu et al. 2023a]. As their names imply, interpolation

Authors' addresses: Akanksha Dixit, Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, Delhi, India, Akanksha.Dixit@ee.iitd.ac.in; Smruti R. Sarangi, srsarangi@cse.iitd.ac.in, Hi-Tech Robotics and Autonomous Systems Chair Professor, Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, Delhi, India.

ACM XXXX-XXXX/2025/9-ART
https://doi.org/XXXXXXX.XXXXXXX

predicts a frame using both past and future frames, whereas extrapolation creates a new frame by utilizing only the past few frames. Fig. 1 shows the performance of a few recent works in terms of quality and runtime. It is evident that interpolation yields superior quality than the existing extrapolation methods but comes with a higher runtime (**almost 14 ms**), whereas extrapolation offers a lower runtime at the expense of inferior quality. This is because interpolation takes into account both past and future frames (see Section 2.1 for more details).

Therefore, the challenge is clear: to achieve the visual quality of interpolation while maintaining the runtime efficiency of extrapolation.

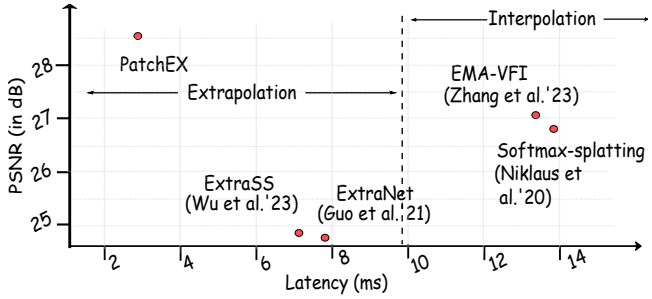


Fig. 1. The solution space for frame generation at 720p. Each solution is run on an NVIDIA RTX 4080 GPU. The detailed system configuration is shown in Table 3.

Note that interpolation introduces an inherent latency by holding an already rendered frame for a refresh interval before displaying it. This is something that the human visual system can easily detect[Dixit and Sarangi 2024]. We thus propose an extrapolation-based approach that does not incur this overhead. Given that historically such algorithms have produced low-quality outputs, real-time extrapolation is a less explored area. As per our knowledge, there are only three major works that specifically address this: ExtraNet [Guo et al. 2021], ExtraSS [Wu et al. 2023a] and STSS [He et al. 2024]. These methods use a warping algorithm[Zhang et al. 2003] that transforms the frame using a motion vector. Warping algorithms often lead to invalid pixels and holes in certain regions and incorrect shading in other regions. Various approaches such as using neural networks have been employed to rectify these issues. They use the information stored in G-buffers – these are data structures in the rendering engine that store different properties of a scene such as the scene depth, roughness, etc.. Despite these efforts, none of the methods have produced satisfactory results in complex dynamic environments with multiple characters and lighting sources. Even though baseline extrapolation methods are fast, the moment neural networks are added, they become very slow.

In this paper, we introduce *PatchEX*, a novel approach that is significantly different from prior frame extrapolation methods. To the best of our knowledge, our work is the first to explicitly incorporate perceptual sensitivity as a fundamental design principle in frame extrapolation. This is a crucial consideration, as foreground regions typically exhibit significantly more parallax than background regions. This parallax difference leads to pronounced occlusion and disocclusion effects that require special attention. Unlike previous works that typically apply uniform processing across the entire frame, potentially ignoring critical content while focusing on regions of less importance, our method segments the frame into foreground and background regions, taking into account the idiosyncrasies of the human vision system (known as *foreground-background segmentation*) [Björkman and Eklundh 2005] and processes these regions *differently*.

Our extrapolation pipeline begins by warping the frame F_t using the current motion vector, which acts as an initial prediction for extrapolation for the extrapolated frame $F_{t+0.5}$. As discussed earlier, the warped frame may have invalid pixels in the disoccluded regions and incorrect shading. Hence, we propose a novel neural network to fix this. We generate two binary masks: one to identify disoccluded regions requiring inpainting, and another for foreground-background separation. These masks are then provided as an input to the neural network designed to repair invalid pixels and correct shading. Our neural network leverages deformable convolution, allowing the model to adaptively focus on content-specific transformations. One more point to note is that previous extrapolation methods rely on multiple G-buffers for warping, marking invalid pixels, and fixing those pixels using neural networks. However, these G-buffers are expensive to generate and are often incompatible with forward rendering in gaming and VR engines. To address this, our approach uses only scene depth and motion vectors, making it lightweight and suitable for both forward and deferred rendering pipelines. Additionally, we incorporate a wavelet transform-based filter pruning strategy for the proposed neural network, enabling the network to maintain high performance with reduced computational cost and runtime.

Currently, we lack large-scale publicly available datasets or workloads for characterizing real-time rendering in graphics applications. To address this, we created a dataset by downloading model and scene files from *Epic Games* [Games 2023a] and rendering them using *Unreal Engine* (v5.1) [Games 2023b]. Our dataset includes multiple animation sequences featuring a diversity of characters, lighting effects, background scenes and camera motions. Note that our contributions are generic and are not specifically limited to our chosen evaluation framework.

To summarize, our primary contributions are as follows:

❶ We propose a lightweight frame extrapolation pipeline that does not rely on expensive and rendering-mode-specific G-buffers.
❷ We propose a novel perceptually guided neural network that processes foreground and background regions differently using deformable convolutions.
❸ We meticulously curated a comprehensive dataset featuring a wide range of animation sequences encompassing diverse characters, backgrounds, lighting settings and camera movements.
❹ To reduce computational cost and runtime, we propose a novel and bespoke filter pruning technique that removes less important filters from the neural network. This makes the network smaller and faster while still producing high-quality results.
❺ *PatchEX* shows an improvement of 61.32% and 49.21% in the PSNR (peak signal-to-noise ratio) compared to the two most recent extrapolation methods, *ExtraNet* and *ExtraSS*, respectively.
❻ The proposed inpainting network is 5× and 2× faster than the

Table 1. Conditions and resulting presentation latency for interpolation and extrapolation techniques with associated latency constraints.

$\forall i, R_i > D$	assumption
$P_i = 3D - R_i$ $R_i + I \leq 2D$	interpolation
$P_i = 0$ $R_{i+1} + E \leq 2D$	extrapolation
R, P: Rendering and presentation latency D: Refresh Interval	

corresponding inpainting networks in the nearest competing works *ExtraNet* and *ExtraSS*, respectively.

The paper is organized as follows. Section 2 describes the background and related work in the area of various frame generation techniques. Section 3 characterizes the datasets and provides the motivation for our proposed approach. Section 4 presents the methodology in detail. The implementation details are provided in Section 5. Section 6 shows the experimental results, and we finally conclude in Section 7.

2 BACKGROUND AND RELATED WORK

2.1 Real-Time Frame Generation

Recent works primarily focus on ❶ predicting new frames using interpolation [Andreev 2010; Herzog et al. 2010; Nehab et al. 2007] and ❷ generating new frames using extrapolation [Guo et al. 2021] to increase the frame rate.

As mentioned in Section 1, apart from the runtime of the algorithm used for interpolation, there is an additional delay incurred here because *interpolation* predicts frames between two already rendered neighboring frames. We thus need to wait more. In contrast, extrapolation-based methods predict frames solely based on past frames. The difference can be observed in Fig. 2. Both processes double the frame rate by generating a new frame after each rendered frame. However, interpolation increases the display or presentation latency. In the figure, the presentation latency is the delay between the completion of a frame's rendering and its actual display on the screen.

The mathematical formulae for the presentation latency of interpolation and extrapolation, respectively, are shown in Table 1 (keep referring to Fig. 2). The first assumption is that the rendering time for every frame is greater than one refresh interval D (in the super-sampled case). If this is not the case, than there is no need to interpolate or extrapolate in the first place. It is further assumed that the rendering duration plus the interpolation/extrapolation time does not exceed two refresh intervals $2D$. We observe in Fig. 2 that if the sum exceeds $2D$, then the interpolated frame will simply not be ready by the time that it needs to be displayed. The assumption here is that we are supersampling by a factor of 2×.

P_i and R_i denote the presentation latency and rendering latency for the i^{th} frame, respectively. D represents the refresh interval. E is the latency associated with generating an extrapolated frame (the $(i + 0.5)_{th}$ frame) based on frame F_i, while I is the latency for generating an interpolated frame using frames F_i and F_{i+1}. If we consider a 90 Hz display, the refresh interval D is 11.11 ms. Hence, the presentation latency P_i for interpolation falls within the range of 11.11 ms to 22.22 ms; this is considerably larger than the latency for extrapolation, which is 0 (in a system without slack).

This latency introduced by interpolation significantly affects the user experience due to the human visual system's acute sensitivity to delays.

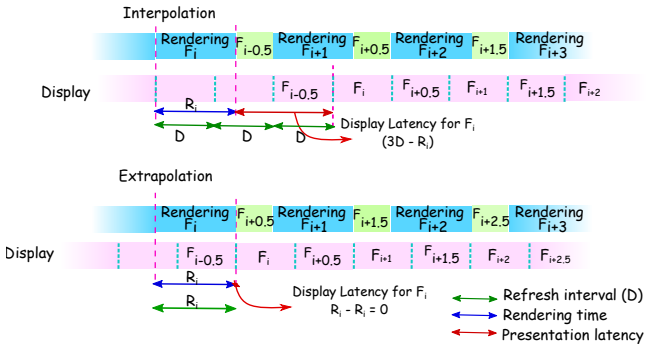


Fig. 2. Interpolation and extrapolation explained. F_i is the rendered frame. R and D represent the rendering time and refresh latency, respectively.

We present a brief comparison of related work in Table 2.

2.1.1 Interpolation. Early frame generation methods used optical flow-guided interpolation [Nehab et al. 2007], but they produced subpar results when the scene contains areas visible in the current frame but not in the previous one. Although Bowles et al. [Bowles et al. 2012] proposed to fix this using an iterative method called fixed point iteration (FPI), this did not provide satisfactory results. To handle this case, various works [Burnes and C Lin 2023; Mueller et al. 2018] propose a bidirectional reprojection method that temporally upsamples frames by reusing data from both the backward and forward temporal directions. For example, NVIDIA's latest DLSS3 engine does this using an optical flow generator, a frame generator and a supersampling network that is AI-accelerated and integrated into its latest GPU architecture (Ada Lovelace [Burnes and C Lin 2023]). This approach increases the frame rate but also leads to an increased input latency that users can easily perceive (verified in the lab and reported in the literature[Guo et al. 2021]). Since our approach is not based on optical flow fields, it does not require future frames to predict a new frame. Other methods, such as caching techniques [Nehab et al. 2007] and dividing frames into slow-moving and fast-moving parts and rendering each part at a different rate [Andreev 2010] have also been proposed but they increase the time needed to construct a frame significantly. Recently, DNN-based solutions [Niklaus and Liu 2020; Zhang et al. 2023] have also been proposed to produce high-quality interpolated frames. However, this increases the latency of the interpolation process due to the complex structure of neural networks.

2.1.2 Extrapolation. Recent frame extrapolation methods designed for desktop applications are ExtraNet [Guo et al. 2021], ExtraSS [Wu et al. 2023a], and STSS [He et al. 2024]. To predict a new frame, all of these works first propose a warping algorithm that helps

Table 2. A comparison of related work

Year	Work	Coherence Exploited	Method Used	ML-based	Real-time	Upsampling Domain	Upsampling Factor
2007	Nehab et al. [Nehab et al. 2007]	Spatio-temporal	Interpolation	×	×	Temporal	
2010	Andreev et al. [Andreev 2010]	Temporal	Interpolation	×	×	Temporal	x to 60
2010	Herzog et al. [Herzog et al. 2010]	Spatio-temporal	Interpolation	×	×	Temporal	
2012	Bowles et al. [Bowles et al. 2012]	Temporal	Interpolation	×	×	Temporal	
2018	SAS [Mueller et al. 2018]	Temporal	Interpolation	×	×	Temporal	x to 120
2020	Softmax-splatting [Niklaus and Liu 2020]	Temporal	Interpolation	✓	×	Temporal	up to 2
2021	ExtraNet [Guo et al. 2021]	Temporal	Extrapolation	✓	✓	Temporal	up to 2×
2022	DLSS 3 [Burnes and C Lin 2023]	Spatio-temporal	Interpolation	✓	✓	Temporal	up to 4×
2023	EMA-VFI [Zhang et al. 2023]	Temporal	Interpolation	✓	×	Temporal	up to 4×
2023	ExtraSS [Wu et al. 2023a]	Temporal	Extrapolation	✓	✓	Spatio-temporal	up to 2×
2024	STSS [He et al. 2024]	Temporal	Extrapolation	✓	✓	Spatio-temporal	2×
2025	***PatchEX***	**Temporal**	**Extrapolation**	✓	✓	**Temporal**	**2×**

generate a warped frame that serves as an initial prediction. However, the warped frame may have invalid pixels or holes in disoccluded regions where the temporal information is not available; this leads to incorrect shading (shadows and reflections). To fix the holes, they use a neural network similar to an image inpainting network [Bertalmio et al. 2000; Guillemot and Le Meur 2013]. However, it may not be sufficient in cases where the shading information changes dynamically over time. To handle this, ExtraNet [Guo et al. 2021] uses a history encoder to learn the shading pattern from the previous few frames and fix that in the warped frame. This approach works well when the lighting conditions change slowly over time.

On the other hand, STSS [He et al. 2024] uses light source information along with a history encoder. ExtraSS [Wu et al. 2023a] introduces a new warping method to minimize the presence of invalid pixels in the warped frame. Their technique utilizes G-buffers' information for the warping process. The G-buffer is a set of render targets that store various properties of a scene during the rendering process. It then uses a lightweight neural network to fine-tune the shading. Sadly, none of these works produce a satisfactory result in a complex dynamic environment with numerous characters and lighting sources. Apart from the quality issue, the runtime of these methods is significant owing to the presence of large neural networks. Note that the latest work ExtraNet [Guo et al. 2021] is a frame generation approach that purely inserts frames in the temporal domain, while the remaining two [He et al. 2024; Wu et al. 2023a] propose a joint neural network for supersampling in both temporal and spatial domains. We focus our research solely on the temporal domain. *We can use a complementary spatial supersampling method if there is a need to increase the resolution of our generated images.*

Let us now introduce some background techniques that we will use in our solution.

2.2 Image Warping

Image warping is a technique that changes the shape or appearance of an image by applying a spatial transformation. There are two types of warping techniques: forward warping and backward warping [Lee et al. 2018; Shimizu et al. 2022]. In forward warping, each pixel in the frame that needs to be warped (the source frame) is directly mapped to a corresponding position in the warped frame using motion information. The mapping is not necessarily injective or surjective. On the other hand, backward warping produces a surjective mapping (there will be no holes in the future frame). It uses an inverse function (the backward map), which is computed by traversing each pixel in a reference target frame and finding its corresponding position in its source frame [Zhang et al. 2003]. This map is subsequently used to perform frame interpolation in a forward direction. Clearly, forward warping is simpler than backward warping, since it directly uses the transformation function to map source pixels to the target. There can be holes because the mapping is not surjective. Backward warping, which is more time consuming, does not have this issue and thus produces better images [Guo et al. 2021; Lee et al. 2018]. This is because of its guarantee of producing a surjective mapping and the fact that its implementations are typically quite efficient.

Using motion vectors results in residual frames or trails of moving objects in the warped frame, also known as the *ghosting* effect. To address this issue, Zeng et al. [Zeng et al. 2021] have proposed a method for generating occlusion motion vectors. These vectors calculate displacements in disoccluded regions as displacements of nearby regions in the previous frame. But it still fails when the background becomes complex. To overcome this challenge, Wu et al. [Wu et al. 2023a] introduced a novel technique called G-buffer-guided warping. Specifically, it considers a large area of pixels near the warped pixel and uses weighted G-buffers' values to blend them to form the warped pixel. However, these G-buffers are expensive to generate and not available in the forward rendering mode. To address this limitation, our method, *PatchEX* avoids reliance on such buffers and instead utilizes only scene depth and motion vectors. Specifically, it uses the occlusion-aware motion vector-based warping, followed by a dedicated inpainting and shading correction network to eliminate visual artifacts and improve frame quality. We have characterized the warping methods mentioned above in Appendix A.1.

2.3 Foreground Bias Effect in Human Vision

The human visual system is an incredibly complex and sophisticated mechanism responsible for perceiving and interpreting visual information. The process of generating new frames can be challenging

Table 3. Platform Configuration

Parameter	Type/Value
CPU	Intel®Xeon®Gold 6226R @ 2.90GHz
GPU	NVIDIA RTX™4080
Game engine	Unreal Engine v5.1

due to the intricate nature of this system, which is highly sensitive to even the slightest input delay and jitter [Ng et al. 2012; Weier et al. 2017]. However, certain characteristics of the human vision system can be leveraged to optimize the frame generation process. One of these characteristics is known as "*foreground bias*" or "*foreground dominance*" [Fernandes and Castelhano 2021]. This phenomenon occurs because the human visual system tends to focus more on objects in the foreground than those in the background. The primary cause is that the foreground objects are usually closer to the observer than background elements, this leads to a greater disparity in the retinal image size. It also provides stronger depth cues. Our visual system is highly sensitive to depth cues, which contribute to the perceptual salience of foreground objects compared to background environments. The foreground bias effect can be used to our advantage by extrapolating foreground interactions more efficiently since artifacts are more noticeable in the foreground as compared to the background.

3 CHARACTERIZATION AND MOTIVATION

In this section, we begin by presenting the benchmarks used in our experiments. We then demonstrate how any frame can be segmented into two distinct regions: foreground and background. Next, we evaluate the performance of various warping methods. We then present the challenges associated with the frame extrapolation methods.

3.1 Overview of the Datasets

To ensure the generalizability of our approach, we aimed to create a large and diverse dataset. We gathered 13 background environments from the UE Marketplace, each with unique artistic styles and complexities. In these environments, we randomly inserted over 20 different characters along with 30 animation sequences, ranging from simple walks to complex hip-hop dances. We then selected good viewpoints and created camera paths to follow the main animation character for each animation sequence to create a variety of sequences. In this regard, we followed the method used to create datasets in recent works [Li et al. 2022; Shugrina et al. 2019] (refer to Appendix A.2 for more details on the dataset). The details of the benchmark applications are shown in Table 4, and sample scenes from a few of the applications are shown in Fig. 3.

We render our datasets using Unreal Engine 5 (UE5 v5.1) on an NVIDIA RTX 4080 GPU with the Ada Lovelace architecture. The detailed configuration is shown in Table 3. To create the animation sequences, we downloaded Unreal scene files from the UE Marketplace [Games 2023a]. We further complicated it by integrating animations with characters from Mixamo [mix 2024] into the background scenes to generate various animation sequences.

Table 4. Graphics benchmarks

Abbr.	Name	Abbr.	Name
PR	City Park	*LB*	Lab
WT	Western Town	*BK*	Bunker
RF	Redwood Forest	*TR*	Tropical
CM	Cemetery	*VL*	Village
BR	Bridge	*TN*	Town
DW	Downtown West	*SL*	Slum
TC	Tennis Court		

3.2 Foreground-Background Segmentation

As mentioned in Section 2.3, the foreground bias effect can be used to our advantage by extrapolating foreground interactions more intricately since artifacts are more noticeable in the foreground as compared to the background. To achieve this, we segment a frame into two parts: foreground and background. We can then use different extrapolation algorithms for each part and blend the outputs.

Insight: Extrapolation can be performed more efficiently by exploiting the fact that humans perceive different parts of an image with varying levels of sensitivity.

3.3 Challenges in Frame Extrapolation

Our primary objective is to predict an intermediate frame called $F_{t+0.5}$ by utilizing the previously rendered frame, F_t. We can generate an estimate of $F_{t+0.5}$ using warping. It will have a lot of visual artifacts primarily because of the presence of disoccluded regions. Note that there are four types of occlusions: self, object-to-object, object-to-background and static. This means that there are four types of disoccluded regions as well (refer to Fig. 4). Self-occlusion occurs when an object obstructs itself in the image. Object-to-object occlusion happens when two or more objects overlap, object-to-background occlusion occurs when an object is partially or wholly occluded by the background, and static disocclusion occurs due to camera movement. These disoccluded regions remain a challenge to fill in making the extrapolation very challenging.

The other challenge in the extrapolation task is accurately predicting changes in *shadows*. Even minor changes in the movement of a dynamic object can result in significant changes in the shadow it casts, as illustrated in Fig. 5. This can have a significant impact on the overall realism of graphics applications, as shadows play a crucial role in conveying depth and dimensionality.

Insights:

❶ The presence of disoccluded regions (❒) and sudden changes in the shadows (⬬) pose a significant challenge for frame extrapolation.
❷ We rely on a novel neural network to fix these issues. It uses different inputs for foreground and background regions.

4 METHODOLOGY

In this section, we propose a formal definition of the frame extrapolation problem before detailing our methodology. Our primary objective is to predict the intermediate frame $F_{t+0.5}$ based on the previous frame F_t and some additional information. In simple terms, we aim to generate a frame that visually sits halfway between two consecutive frames while ensuring coherence with the overall sequence

(a) Bunker (b) City Park (c) Western Town (d) Redwood Forest

(e) Cemetery (f) Bridge (g) Downtown West (h) Tennis Court

Fig. 3. Example views from a few sample scenes

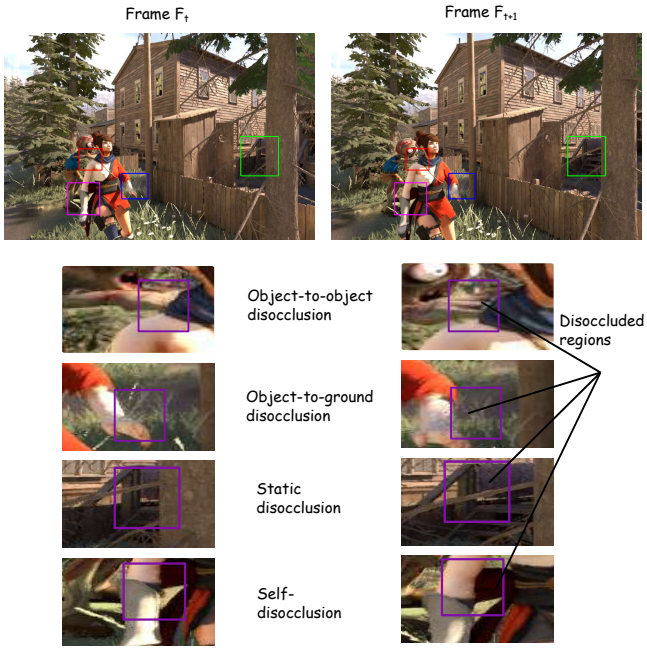
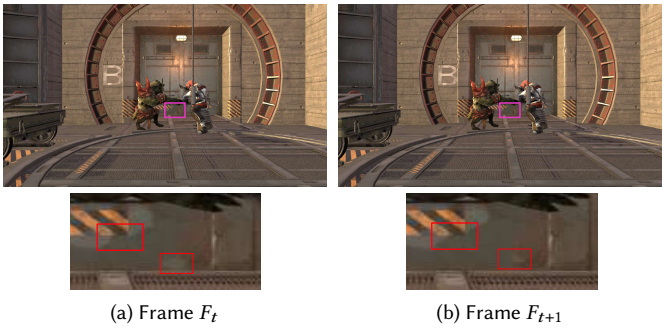


Fig. 4. Disoccluded regions in a frame



(a) Frame F_t (b) Frame F_{t+1}

Fig. 5. Dynamic changes in the appearance of shadows between two successive frames, F_t and F_{t+1}.

of frames. Our proposed method, *PatchEX*, for frame extrapolation comprises two stages: ❶ mask generation, and ❷ inpainting and shading correction. The goal is to accurately predict future frames by handling disocclusion and appearance distortions introduced during warping.

(1) **Mask Generation: Segmentation and Disocclusion** In the first step, we identify regions that require inpainting due to disocclusions. We achieve this by generating two types of masks: disocclusion mask (for the disoccluded regions) and segmentation mask (foreground-background separation). To compute these masks, we utilize depth information and motion vectors extracted from the rendering pipeline. Both of them are binary masks.

(2) **Inpainting and Shading Correction Network** The second stage is a neural network designed to inpaint the disoccluded regions and apply shading corrections to produce a temporally coherent and photorealistic extrapolated frame. This is the crux of our contribution.

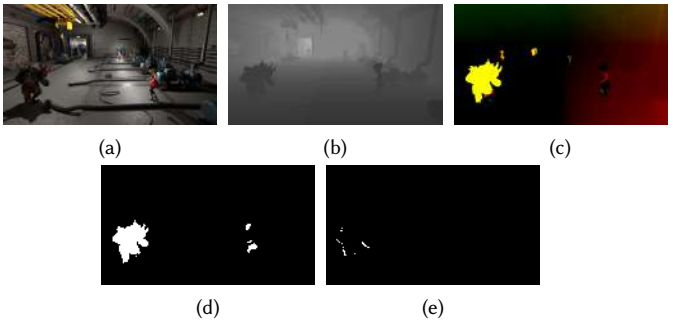
In the following section, we provide a detailed discussion of *PatchEX*.

5 IMPLEMENTATION

To realize our two-stage frame extrapolation pipeline after warping, we extract auxiliary buffers directly from the rendering pipeline for each frame: scene depth (used in mask generation), pretonemapped color (used in warping), and motion vectors (used in mask generation and provided as an input to the neural network). We use these G-buffers for mask generation and motion-aware extrapolation. A visualization of all these buffers is shown in Fig. 6.

(a) (b) (c)

Fig. 6. (a) Scene depth; (b) PretonemapHDRColor; (c) Motion vector



(a) (b) (c) (d) (e)

Fig. 7. (a) Frame F_t; (b) Scene depth; (c) Motion vector; (d) Foreground mask; (e) Disocclusion mask.

5.1 Mask Generation: Segmentation and Disocclusion

Segmentation Mask: In this section, we divide the frame into the foreground and background regions by creating a binary mask. Foreground detection is in general a complex task in video processing, but with access to some auxiliary information, we can efficiently separate the foreground from the background. The motion vectors encode the displacement of each pixel between the frames F_{i-1} and F_i. Since the camera may also be moving, we compensate for global motion by subtracting the average displacement from the motion vectors. We classify a pixel as foreground if it meets two conditions: (1) it moves at a speed greater than a predefined threshold, and (2) its depth falls within a specified range. This method provides an efficient and adaptive way to segment the scene, reducing reliance on manually generated masks while improving accuracy in handling occlusion and disocclusion. To decide the threshold values, we perform a sensitivity analysis in Section 6.8.

Disocclusion Mask: To detect regions that become newly visible in a frame due to moving objects (and not just camera motion), we use scene depth and motion vector data exported from the rendering pipeline. We start by *aligning* the scene depth map from the previous frame with the current frame's viewpoint. To do this, we warp the previous depth map so that it matches the perspective of the current frame. After aligning the depth maps, we subtract the warped depth map from the current depth map. If the difference in the depth at a pixel is greater than a threshold (=0.1), we consider that pixel to be disoccluded. We generate a binary mask where these disoccluded pixels are marked. This mask highlights the regions that were hidden in the last frame but are now visible, and is later used to guide the extrapolation network in focusing on these newly revealed (disoccluded) areas. Fig. 7 shows an example of foreground and disocclusion masks.

5.2 Inpainting and Shading Correction Network

Following the generation of both segmentation and disocclusion masks, we employ a custom-designed inpainting and shading correction neural network to generate the final extrapolated frame.

5.2.1 Neural Network Architecture. We propose a novel, bespoke neural network for the extrapolation task. Its job is to take the masks, the warped frame and remove all visual artifacts. The architecture of the network is shown in Fig. 8. Our network is inspired by the classical encoder-decoder architecture. In our design, the encoder and decoder comprise three hierarchical stages. Each stage in the encoder consists of a downsampling operation, followed by a convolutional block composed of a convolutional layer, batch normalization and a ReLU activation. The decoder mirrors this structure, with each stage performing an upsampling operation followed by a similar convolutional block. However, instead of using conventional convolutional layers, we replace them with *Lightweight Gated Convolution* layers. These layers adopt the efficient gated convolution mechanism introduced by Yi et al. [Yi et al. 2020] (refer to Equation 1). The network was chosen because it can efficiently subtract the background, amplify salient structures, nicely track object boundaries and has a fast implementation. To the best of our knowledge, this is a novel network where we have skip connections with two destinations and one skip connection is interjected by a deformable convolution layer.

$$\begin{aligned} G(gating) &= Conv(W_g, I) \\ F(features) &= Conv(W_f, I) \\ O &= \sigma(G) \odot F \end{aligned} \tag{1}$$

W_g and W_f denote two distinct learnable filters. $\odot$ denotes Hadamard (element-wise) multiplication, and σ represents the sigmoid activation function. The latter ensures that the output gating values are in the range $[0, 1]$. This approach helps in treating different pixels differently in the network since there are invalid pixels in the warped frame. Our gating mechanism diminishes the influence of invalid pixels (holes).

Additionally, to enhance spatial adaptability in regions of complex motion or occlusion, we incorporate deformable convolutions as proposed by Dai et al. [Dai et al. 2017]. Specifically, we insert deformable convolution layers at the first stage of the encoder and before the final stage of the decoder. These layers enable the network to learn large and flexible receptive fields (refer to Fig. 9), allowing for more precise feature alignment in challenging regions of the frame. Please refer to Appendix A.5 for seeing how deformable convolution is different from the standard convolution.

5.2.2 Inputs to the Network. We provide a comprehensive set of inputs: the warped frame, a hole mask indicating disoccluded regions marking invalid pixels, one G-buffer (motion vector) and one binary mask for foreground regions. Unlike previous works, we also provide as input the LBP (local binary pattern) feature map of the warped frame. It computes an LBP code for each pixel, which

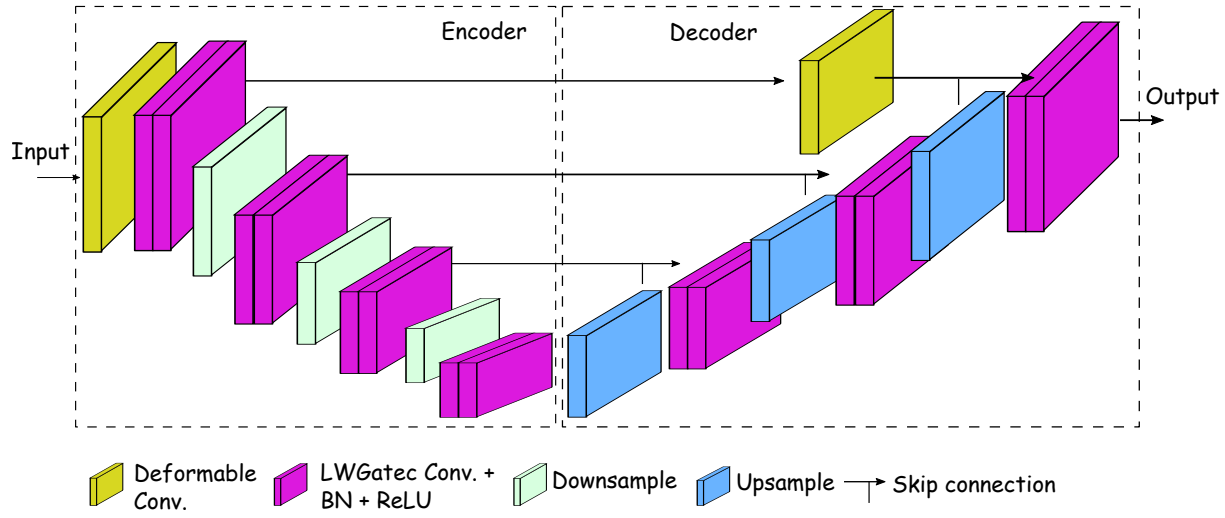


Fig. 8. The neural network architecture for fixing the invalid pixels.

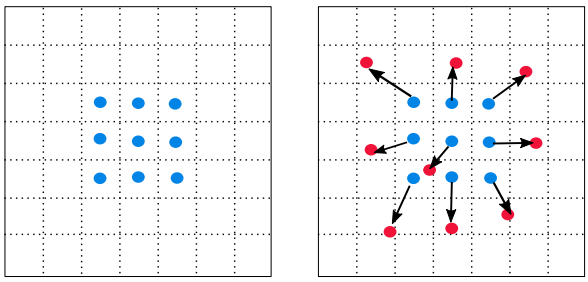


Fig. 9. Illustration of the sampling locations using 3 × 3 standard and deformable convolutions. Left figure: regular sampling grid (blue points) of standard convolution. Right figure: deformed sampling locations (dark red points) with augmented offsets (black arrows) in deformable convolution.

can help us further classify it into the following categories: flat region, bright spot, edge, corner, etc. By incorporating the LBP feature map, we leverage its robust feature extraction capabilities and its resilience to uneven illumination. This addresses the challenge of extracting detailed features from images with inconsistent lighting, significantly boosting the generalizability and effectiveness of our method.

5.2.3 Loss Functions. The *loss function* used for training the networks has broadly two components (refer to Equation 2). The first component penalizes the pixel-wise error between the ground truth F and the predicted frame F'. The second component is the perceptual loss, which was not considered in previous works [Guo et al. 2021; He et al. 2024]. The *perceptual loss* plays a crucial role in enhancing the performance of neural networks, particularly in tasks related to shading correction[Ran et al. 2023] because it focuses on capturing high-level perceptual features, mimicking human visual perception.

$$\mathcal{L} = \mathcal{L}_{pixel} + \mathcal{L}_{perceptual} \tag{2}$$

Pixel-wise Errors: To calculate the error on a pixel-by-pixel basis, we use the $\mathcal{L}_1$ loss metric, which can be computed using the formula shown in Equation 3. We employ the $\mathcal{L}_1$ loss in two different forms. First, we calculate the total $\mathcal{L}_1$ loss between the entire ground truth frame and the predicted frame. Next, we calculate the error between the pixels that were marked as holes or invalid pixels during using the disocclusion mask. This loss is designed to emphasize more on invalid regions. The total loss is the weighted sum of these two losses (refer to Equation 4).

$$\begin{aligned} \mathcal{L}_{\mathcal{L}_1} &= \|X - Y\|_1 \\ \mathcal{L}_{\mathcal{L}_1} &= \sum_{i=1}^{H}\sum_{j=1}^{W}|X(i,j) - Y(i,j)| \end{aligned} \tag{3}$$

$$\begin{aligned} \mathcal{L}_{\mathcal{L}_1} &= \|F - F'\|_1 \\ \mathcal{L}_{hole} &= \|(F - F') \cdot (1 - m)\|_1 \\ \mathcal{L}_{pixel} &= \lambda_{\mathcal{L}_1} \cdot \mathcal{L}_{\mathcal{L}_1} + \lambda_{hole} \cdot \mathcal{L}_{hole} \end{aligned} \tag{4}$$

m is a binary mask used for identifying the invalid pixels. $\lambda_{\mathcal{L}_1}$ and λ_{hole} are the weights assigned to each component loss function to balance out their effects. In our current implementation, $\lambda_{\mathcal{L}_1}$ and λ_{hole} are set to 1 and 0.4, respectively.

Perceptual Losses: We adopt the perceptual loss function proposed by Johnson et al. [Johnson et al. 2016], which comprises two components: VGG loss and style loss. These components jointly guide the neural network to generate images that better align with human visual perception. $\mathcal{L}_{VGG}$ is the VGG loss (shown in Equation 5). Here, Φ_i is the activation map of the i^{th} layer of the VGG-16 [Liu and Deng 2015] network pre-trained on ImageNet.

$$\mathcal{L}_{VGG} = \mathbb{E}\left[\sum_i \|\Phi_i(F) - \Phi_i(F')\|_1\right] \tag{5}$$

We use another loss function called *style loss* to maintain a degree of similarity between the predicted and the original image (refer to Equation 6). The joint perceptual loss is shown in Equation 7. The

Table 5. Statistics of the training and testing dataset

Scenes	Training Sequences	Training Frames	Testing Sequences	Testing Frames
PR	5	5000	3	3000
WT	5	5000	3	3000
RF	5	5000	3	3000
CM	5	7000	3	3000
BR	5	5000	3	3000
DW	5	4000	3	3000
TC	5	4000	3	3000
TN	5	5000	3	3000
BK	0	0	3	3000
LB	0	0	3	3000
TR	0	0	3	3000
VL	0	0	3	3000
SL	0	0	3	3000

key idea is that we are preventing *hallucination*, where the output of the model can be very different from the original image. The main aim is to fix the base image created by warping and not make unconstrained errors.

$$\mathcal{L}_{style} = \mathbb{E}\left[\sum_i \|G_j^{\Phi_i}(F) - G_j^{\Phi_i}(F')\|_1\right] \tag{6}$$

$G_j^{\Phi_i}(F)$ is the Gram matrix computed from the j^{th} feature map of the i^{th} layer of the VGG network (Φ_i).

$$\mathcal{L}_{perceptual} = \lambda_{VGG} \cdot \mathcal{L}_{VGG} + \lambda_{style} \cdot \mathcal{L}_{style} \tag{7}$$

λ_{VGG} and λ_{style} are the weights assigned to each loss to balance their effects. In our current implementation, λ_{VGG} is set to 0.1 and λ_{style} is set to 0.01.

5.2.4 Training and Testing. In our work, we utilize 13 benchmark scenes as discussed in Section 3.1. These scenes are divided into two groups: nine for training and four for testing. For each scene, we created five animation sequences for training and three sequences for testing. The number of frames in each sequence varies. The total number of frames for training and testing for each scene are specified in Table 5. We implemented the neural network using the PyTorch framework [Paszke et al. 2019]. To divide the data for training and validation in an 80:20 ratio, we utilized PyTorch's inbuilt *random_split* function. We use the Adam optimizer for optimization with batch sizes set to 16 and epoch sizes to 100. The network initialization is as per the default settings in PyTorch.

5.3 Neural Network Compression

We propose a filter pruning technique to remove filters that correspond to feature maps with low importance. This compresses the size of the network and makes its implementation more efficient. Our pruning strategy is guided by the region-specific perceptual relevance of features and is motivated by the observation that foreground regions typically contain high-frequency content, while background regions are dominated by low-frequency components. To capture this distinction, we apply a wavelet transform to the feature maps [Liu et al. 2024], which allows us to decompose them into frequency bands while preserving spatial information. Unlike other frequency transforms, the wavelet transform provides a localized view of both high- and low-frequency features, making it well-suited for tasks that require spatial awareness—such as distinguishing foreground from background. We then compute the energy in these frequency bands to derive a perceptual importance score for each filter. Filters contributing primarily to low-importance (typically background) regions are pruned.

Assume that we have a CNN with L convolution layers and the ℓ^{th} convolution layer C^ℓ has N_ℓ filters. The output feature map of this layer can be represented as $F^\ell = \left\{f_1^\ell, f_2^\ell, \ldots, f_j^\ell, \ldots, f_N^\ell\right\}$. If we compute the wavelet transform (WT) of the j^{th} feature map f_j^ℓ, it yields four frequency domain components: the low-frequency component LL, the horizontal high-frequency component HL, the vertical high-frequency component LH and the diagonal high-frequency component HH. We set $LF = LL$ and $HF = LH + HL + HH$. Hence, the wavelet transform map of the output feature map F_ℓ of the ℓ^{th} convolution layer can be represented as:

$$WT^\ell = \left\{(LF_1^\ell, HF_1^\ell), (LF_2^\ell, HF_2^\ell), \ldots, (LF_j^\ell, HF_j^\ell), \ldots, (LF_{N_\ell}^\ell, HF_{N_\ell}^\ell)\right\} \tag{8}$$

where (LF_j^ℓ, HF_j^ℓ) represents the WT of the j^{th} feature map of the ℓ^{th} convolutional layer. This can also be written as $WT(f_j^\ell)$. We then compute the energy of each feature map in both frequency bands as follows:

$$\begin{aligned} E_j^{\ell,LF} &= \sum_{m=1}^{H_\ell}\sum_{n=1}^{W_\ell} LF_j^\ell(m,n)^2 \\ E_j^{\ell,HF} &= \sum_{m=1}^{H_\ell}\sum_{n=1}^{W_\ell} HF_j^\ell(m,n)^2 \end{aligned} \tag{9}$$

Here, $E_j^{\ell,LF}$ and $E_j^{\ell,HF}$ refer to the energy of the j^{th} feature map for low and high-frequency bands, respectively. We then define a scalar perceptual importance score of the j^{th} feature map that blends the two energy terms using a tunable hyperparameter α (refer to Equation 10).

$$imsco(f_j^\ell) = \alpha.\frac{E_j^{\ell,HF}}{E_j^{\ell,HF} + E_j^{\ell,LF}} + (1-\alpha).\frac{E_j^{\ell,LF}}{E_j^{\ell,HF} + E_j^{\ell,LF}} \tag{10}$$

Here, α (=0.8) controls the weight assigned to foreground activity vs. background response. According to previous studies, it is known that the importance scores generated from individual feature maps using wavelet transform are robust to the input images [Lin et al. 2020]. Finally, we sort the filters in ascending order of their importance score and the filters corresponding to the sorted feature maps are pruned. The pruned network is then retrained, and this pruning–retraining process is repeated iteratively for 200 epochs using a batch size of 128.

6 RESULTS AND ANALYSIS

① To evaluate the visual quality of the proposed method, *PatchEX*, we conduct a comprehensive comparison with various state-of-the-art frame extrapolation techniques. Since all these approaches rely on machine learning, we fine-tune their respective neural networks on our dataset before assessing their performance.

② Subsequently, we perform an ablation study to analyze the contribution of individual components to the overall effectiveness of *PatchEX*.
③ Moreover, we compare *PatchEX* against state-of-the-art frame generation techniques based on interpolation.
④ Specifically, we measure the runtime of each component in *PatchEX*.
⑤ To assess robustness, we compare *PatchEX* with VR-style extrapolation scenarios that demand wide field-of-view consistency and perceptual quality.
⑥ We further compare *PatchEX* with commercial extrapolation solution to highlight its practical advantages and limitations.
⑦ We analyze the scalability of *PatchEX* in high-resolution scenarios to verify its performance in more demanding use cases.

For all these experiments, we use the same system configuration (refer to Table 3). Unless otherwise specified, all comparisons are performed at a default spatial resolution of 720p, which is consistent across all experiments. Also, the input frames are rendered at 60 fps from UE. The results of *PatchEX*, along with those of other state-of-the-art approaches, are available at the following link [Authors 2024].

6.1 Comparison with Frame Extrapolation Methods

In this section, we compare the visual performance of ExtraNet [Guo et al. 2021] and ExtraSS [Wu et al. 2023a] with *PatchEX*, both qualitatively and quantitatively. As mentioned in Section 2.1.2, ExtraNet and ExtraSS are the two nearest competing methods that perform frame extrapolation in real-time. However, ExtraSS does not solely extrapolate in the temporal domain; it also extrapolates in the spatial domain. Since we are dealing in the temporal domain not spatial, we only consider its temporal component for the purpose of comparison.

6.1.1 Qualitative Comparisons. In this section, we compare the quality of the frame generated using various extrapolation methods. In Fig. 10, we show the extrapolated frames for four distinct benchmark scenes.

As explained in Section 3.3, the challenges for extrapolation algorithms are to properly fill the disoccluded regions created by the movements in the scene and to extrapolate the shadows accurately. Both ExtraNet and ExtraSS fail to address these challenges effectively in many scenarios. For example, ExtraNet does not generate accurate shadows for complex movements such as the kick in the *RF* scene. It also performs poorly in capturing complex structures such as tree leaves and facial features in scenes such as *BK, RF* and *WT*. On the other hand, ExtraSS leverages G-buffer-guided warping and performs better than ExtraNet in most cases. However, it is unable to correctly extrapolate facial features during intricate motions like those in a hip-hop dance. In the *DW* scene, ExtraSS handles the out-of-screen areas well, whereas ExtraNet does not. *PatchEX* handles all of these complex cases very well (much better than ExtraNet and ExtraSS). To summarize, *PatchEX* excels by not only preserving sharp features and intricate geometries but also generating plausible shadows that closely match the ground truth.

6.1.2 Quantitative Comparisons. In this section, we present a quantitative comparison of *PatchEX*, ExtraNet, and ExtraSS using four performance metrics: PSNR, SSIM [Hore and Ziou 2010], VMAF [Netflix 2016], and LPIPS [Belhe et al. 2023]. We present the full-frame quality assessment in Table 6. We also evaluate the region-wise performance of different frame extrapolation methods by analyzing their PSNR values for foreground, disoccluded, and background regions. This analysis provides insights into how each method performs under varying motion complexities and occlusion patterns. The results are summarized in Table 7. Based on these results, we derive the following conclusions:

❶ *PatchEX* performs better than ExtraNet and ExtraSS across all benchmarks for all the performance metrics.

❷ There is a 61.32%, 33.46%, and 90.16% increase in PSNR, SSIM, and LPIPS in *PatchEX*, respectively, as compared to ExtraNet.

❸ Compared to ExtraSS, *PatchEX* achieves an improvement of 49.21%, 32.29%, and 91.20% in PSNR, SSIM, and LPIPS, respectively.

6.2 Ablation Study

To thoroughly understand the impact of different components in *PatchEX*, we conducted an ablation study focusing on three key elements: foreground-background segmentation, deformable convolution layer and perceptual loss. This analysis helps to isolate the contribution of each component to the overall performance of *PatchEX*. We evaluate three distinct variants of *PatchEX*: one where foreground-background segmentation is not used, another without deformable convolution, and a third without perceptual loss. The quantitative comparison of these variants along with the original method is given in Table 8. We make the following observations from the results:

❶ These results highlight the significant impact of foreground-background segmentation on improving image quality. Without segmentation, the average PSNR decreases by almost 4.14 dB.

❷ Similarly, we see the impact of the deformable convolution layers, there is an improvement of 4.0 dB in PSNR.

❸ Likewise, we observe the significant impact of the perceptual loss we included in the training of neural networks, resulting in an improvement of 1.61 dB in PSNR. These findings remain consistent for SSIM as well. This underscores the effectiveness of incorporating perceptual loss in improving both pixel-level fidelity and structural similarity in the reconstructed frames.

6.3 Performance Comparison with Frame Interpolation Methods

In this section, we compare the performance of *PatchEX* with two state-of-the-art interpolation-based methods. The interpolation-based methods are Softmax Splatting [Niklaus and Liu 2020] and EMA-VFI [Zhang et al. 2023]. Both of these methods are DNN-based techniques. Softmax splatting uses forward warping using optical flow. However, in this approach, multiple pixels may map to the same target location in the frame F_t. Softmax splatting uses a modified softmax layer, which takes the frame's depth data to resolve this ambiguity. EMA-VFI uses a transformer network to perform frame interpolation.

Fig. 10. Visual comparison of the frame extrapolation methods: ExtraNet [Guo et al. 2021] and ExtraSS [Wu et al. 2023a]

6.3.1 Qualitative Comparisons. In this section, we perform a qualitative comparison. It is important to note that while all interpolation methods require pre-rendered future frames, our extrapolation-based method exclusively relies on historical frames that have already been rendered. Still, there are instances where interpolation-based methods exhibit shortcomings. To illustrate such cases, we present frames from three distinct scenes in Fig. 11, emphasizing the visual quality and effectiveness of each approach.

In the *PR* scene, both interpolation-based techniques produce shadows that closely resemble the ground truth. However, in the *BK* scene, the sharp definition of the shadow structure is compromised for both these methods. This can be attributed to the higher glossiness factor present in the *BK* scene as compared to others.

Table 6. Quantitative comparison of various extrapolation methods against *PatchEX* in terms of PSNR (dB), SSIM, LPIPS and VMAF. Throughout this paper, the best and second-best results of each test setting are highlighted in bold red and underlined blue, respectively.

Scenes	PSNR (dB) ↑			SSIM ↑			LPIPS ↓			VMAF ↑		
	ExtraSS	ExtraNet	*PatchEX*	ExtraSS	ExtraNet	*PatchEX*	ExtraSS	ExtraNet	*PatchEX*	ExtraSS	ExtraNet	*PatchEX*
PR	24.21	24.26	**27.18**	0.906	0.899	**0.941**	0.190	0.185	**0.107**	84.15	83.25	**88.69**
BK	28.78	26.87	**31.49**	0.948	0.909	**0.958**	0.111	0.118	**0.106**	87.25	85.36	**91.25**
WT	26.01	23.75	**27.32**	0.848	0.796	**0.898**	0.109	0.213	**0.108**	87.47	83.29	**89.25**
RF	21.55	20.51	**24.55**	0.785	0.733	**0.895**	0.261	0.342	**0.105**	82.35	81.02	**85.01**
CM	23.24	22.97	**24.72**	0.867	0.802	**0.869**	0.226	0.261	**0.117**	83.97	82.05	**85.02**
BR	26.55	24.69	**28.78**	0.814	0.783	**0.887**	0.384	0.465	**0.109**	87.20	85.02	**90.36**
DW	23.03	21.63	**24.55**	0.876	0.791	**0.884**	0.376	0.269	**0.124**	85.20	82.98	**87.58**
TC	23.78	22.36	**26.63**	0.729	0.705	**0.880**	0.398	0.362	**0.122**	83.14	80.14	**86.39**
LB	27.54	24.49	**28.59**	0.846	0.801	**0.889**	0.191	0.414	**0.111**	86.05	84.25	**89.25**
TR	27.86	27.40	**30.92**	0.837	0.877	**0.879**	0.295	0.445	**0.121**	90.25	88.74	**91.19**
VL	22.19	21.00	**26.23**	0.817	0.818	**0.895**	0.178	0.314	**0.123**	83.04	81.25	**88.05**
TN	26.91	25.39	**27.35**	0.854	0.815	**0.905**	0.205	0.305	**0.109**	84.16	85.87	**90.03**
SL	26.47	26.52	**31.19**	0.867	0.810	**0.898**	0.496	0.409	**0.154**	87.25	87.34	**91.58**

Table 7. Region-wise performance comparison of various extrapolation methods against *PatchEX* in terms of PSNR (dB).

Scenes	Foreground			Disoccluded			Background		
	ExtraSS	ExtraNet	*PatchEX*	ExtraSS	ExtraNet	*PatchEX*	ExtraSS	ExtraNet	*PatchEX*
PR	20.80	28.81	**29.87**	**35.73**	26.77	28.36	20.22	20.39	**27.86**
BK	25.00	19.03	**30.25**	26.26	20.90	**28.30**	20.12	21.79	**23.68**
WT	25.54	22.30	**35.47**	**30.88**	29.81	30.25	21.33	25.55	**31.39**
RF	24.52	16.16	**28.47**	**25.89**	22.47	25.47	18.83	18.44	**23.85**
CM	28.77	24.00	**30.14**	30.17	27.52	**30.90**	21.28	19.75	**24.27**
BR	34.01	29.11	**35.40**	21.98	27.21	**31.98**	19.99	19.90	**22.99**
DW	32.30	24.94	**33.12**	25.13	26.30	**30.63**	26.25	24.27	**28.22**
TC	34.53	23.85	**35.24**	26.64	25.68	**34.66**	23.44	22.27	**29.40**
LB	28.29	26.33	**29.29**	27.47	22.58	**32.74**	22.97	27.82	**32.87**
TR	23.19	16.18	**25.14**	26.70	20.18	**30.81**	19.66	18.76	**24.66**
VL	17.62	20.30	**28.25**	26.09	22.94	**32.49**	16.08	16.35	**20.36**
TN	30.78	24.82	**36.25**	25.83	24.23	**34.47**	20.24	21.81	**26.94**
SL	26.13	21.20	**30.01**	23.50	28.89	**33.90**	20.05	19.79	**25.45**

Table 8. Quantitative comparison of various variants of *PatchEX* in terms of PSNR (dB) and SSIM. *w/o FS* refers to without foreground-background segmentation. *w/o DC* refers to the case where the deformable convolution layer is not used. *w/o* $\mathcal{L}_p$ refers to without perceptual loss taken into account.

Scenes	PSNR (dB) ↑				SSIM ↑			
	w/o FS	w/o DC	w/o $\mathcal{L}_p$	*PatchEX*	w/o DC	w/o FS	w/o $\mathcal{L}_p$	*PatchEX*
PR	23.91	21.64	26.61	**27.18**	0.790	0.791	0.892	**0.941**
BK	24.32	25.40	27.50	**31.49**	0.785	0.785	0.885	**0.958**
WT	25.70	21.55	27.71	**27.32**	0.794	0.795	0.884	**0.898**
RF	21.43	20.44	23.43	**24.55**	0.791	0.793	0.893	**0.895**
CM	21.59	20.52	22.54	**24.72**	0.761	0.759	0.860	**0.869**
BR	21.42	23.10	26.44	**28.78**	0.779	0.781	0.879	**0.887**
DW	21.04	22.83	23.82	**24.55**	0.781	0.782	0.882	**0.884**
TC	24.54	20.28	25.91	**26.63**	0.774	0.779	0.877	**0.880**
LB	24.26	26.75	27.01	**28.59**	0.785	0.788	0.888	**0.889**
TR	25.89	28.50	29.61	**30.92**	0.780	0.781	0.881	**0.879**
VL	22.58	23.55	24.98	**26.23**	0.794	0.787	0.896	**0.895**
TN	23.59	25.92	26.05	**27.35**	0.789	0.791	0.892	**0.905**
SL	26.72	27.12	28.23	**31.19**	0.798	0.794	0.897	**0.898**

Notably, *PatchEX* excels in this scenario. Another notable artifact in interpolation methods is the potential for blurriness during complex movements, as observed in the *WT* scene.

6.3.2 Quantitative Comparisons. In addition to the qualitative analysis, we also perform a quantitative comparison of the frame interpolation methods. Table 9 presents the quantitative evaluation in terms of the PSNR, SSIM and VMAF metrics. From these results, we make the following observations:

❶ In terms of PSNR, *PatchEX* consistently outperforms both EMA-VFI and Softmax-splatting across most scenes. Even in cases where it is not the best, the performance gap remains negligible. On average, *PatchEX* achieves a PSNR gain of 1.68 dB over EMA-VFI and 1.21 dB over softmax-splatting.

❷ In terms of SSIM, *PatchEX* demonstrates superior performance in several scenes. This is primarily because interpolation-based methods tend to introduce blurriness, which degrades structural fidelity and leads to lower SSIM values. By contrast, *PatchEX* better preserves fine structural details, resulting in consistently higher SSIM scores.

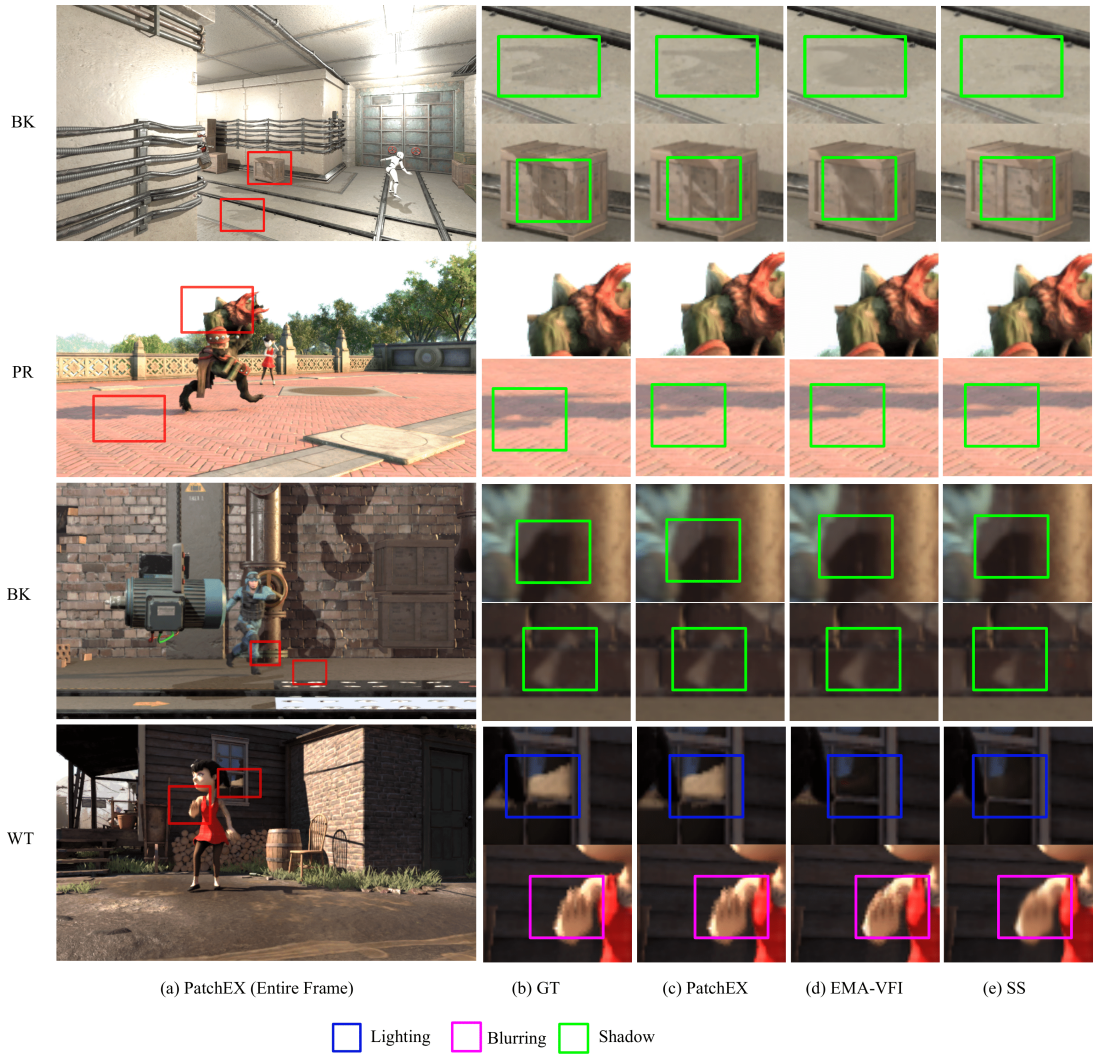


Fig. 11. Visual comparisons against two frame interpolation methods: EMA-VFI [Zhang et al. 2023] and Softmax-splatting (SS) [Niklaus and Liu 2020]

6.4 Runtime Performance of *PatchEX*

As mentioned in Section 4, we divide *PatchEX* into several steps. In Table 10, we report the runtime of each step, measured as the execution time on a single frame. We make the following observations from the table:

❶ Due to the proposed filter pruning for the inpainting and shading correction network, the inference time is very low. Specifically, for 720p resolution, the inference time is just 1.08 ms. In comparison, ExtraNet and ExtraSS have inference times of 5.39 ms and 2.05 ms, respectively. This makes our neural network 5× and 2× faster than ExtraNet and ExtraSS, respectively.

❷ Even for higher resolutions such as 1080p, the inference time of *PatchEX* remains remarkably low, averaging only about 1.24 ms.

❸ For other components, such as warping and mask generation, we observe an expected increase in runtime as resolution increases.

We further evaluate the runtime efficiency of *PatchEX* across different resolution levels w.r.t state-of-the-art methods: ExtraNet and ExtraSS (ExtraSS-E: without spatial supersampling). The time includes all the steps (G-buffers generation, pre-processing, warping and inference). Table 11 shows the runtime (in milliseconds) for all three methods at various resolutions.

❶ The efficiency gain of *PatchEX* becomes more pronounced at higher resolutions. For instance, at 1080p, *PatchEX* runs in 4.01 ms, compared to 9.05 ms for ExtraSS and 15.10 ms for ExtraNet.

❷ The runtime of *PatchEX* increases much more slowly with resolution than both baselines, highlighting its scalability. While ExtraNet's runtime increases fivefold from 360p to 1080p, *PatchEX* only doubles (2.6 ×).

Table 9. Quantitative comparison of various interpolation methods against *PatchEX* in terms of PSNR (dB), SSIM and VMAF.

Scenes	PSNR (dB) ↑			SSIM ↑			VMAF ↑		
	EMA-VFI	SS	*PatchEX*	EMA-VFI	SS	*PatchEX*	EMA-VFI	SS	*PatchEX*
PR	25.05	26.47	**27.18**	0.818	0.809	**0.941**	85.36	86.21	**88.69**
BK	33.56	**34.47**	31.49	**0.958**	0.865	**0.958**	92.50	**93.05**	91.25
WT	26.91	25.62	**27.32**	0.881	0.878	**0.898**	85.31	85.11	**89.25**
RF	23.92	24.11	**24.55**	0.861	0.863	**0.895**	83.14	84.36	**85.01**
CM	23.48	22.22	**24.72**	**0.875**	0.866	0.869	82.36	80.28	**85.02**
BR	27.23	27.73	**28.78**	0.885	0.886	**0.887**	88.21	88.68	**90.36**
DW	24.17	23.88	**24.55**	0.857	0.855	**0.884**	85.07	83.25	**87.58**
TC	**28.27**	27.89	26.63	**0.880**	0.879	**0.880**	**89.36**	88.41	86.39
LB	**29.22**	21.58	28.59	**0.891**	**0.891**	0.889	**90.14**	81.57	89.25
TR	28.74	29.08	**30.92**	**0.889**	**0.889**	0.879	88.27	90.24	**91.19**
VL	26.09	26.14	**26.23**	0.886	0.886	**0.895**	84.21	85.36	**88.05**
TN	**29.59**	28.96	27.35	0.884	0.884	**0.905**	88.97	89.17	**90.03**
SL	29.10	29.74	**31.19**	0.883	0.884	**0.898**	90.21	91.18	**91.58**

Table 10. Runtime (ms) breakdown of *PatchEX* at various resolution levels

Res	Step		
	Warping	Mask-generation	Inference
360p	0.75	**0.34**	0.56
480p	0.79	**0.59**	0.88
720p	0.91	**0.73**	1.08
1080p	1.35	1.41	**1.24**

Table 11. Runtime (ms) comparison with the state-of-the-art methods at various resolution levels

Res	ExtraNet	ExtraSS	*PatchEX*
360p	3.17	2.99	**1.65**
480p	4.11	3.89	**2.26**
720p	7.99	7.08	**2.72**
1080p	15.10	9.05	**4.01**

Table 12. Layer-wise comparison of parameters, inference time, and FLOPs before and after pruning at 720p resolution.

Layer	#Params Before	#Params After	#FLOPs Before	#FLOPs After	Time Before	Time After
DefConv 1	0.23 M	0.17 M	10.87 M	5.99 M	0.089	0.049
LWGatedConv 1	0.54 M	0.14 M	18.15 M	4.26 M	0.149	0.035
LWGatedConv 2	0.54 M	0.48 M	18.15 M	6.79 M	0.149	0.056
DownLWGated 1	1.63 M	0.20 M	15.76 M	7.53 M	0.129	0.062
DownLWGated 2	1.63 M	0.47 M	10.46 M	2.27 M	0.086	0.019
DownLWGated 3	1.63 M	0.20 M	1.81 M	0.55 M	0.015	0.005
UpLWGated 1	1.74 M	0.46 M	29.11 M	5.12 M	0.239	0.042
UpLWGated 2	2.03 M	0.54 M	39.81 M	9.18 M	0.327	0.075
DefConv 2	1.10 M	0.28 M	82.28 M	23.59 M	0.675	0.194
UpLWGated 3	1.56 M	0.46 M	141.89 M	50.94 M	1.264	0.543
Total	12.62 M	3.35 M	368.35 M	131.63 M	3.122	1.080

DefConv: Deformable Conv., **LWGatedConv**: LW Gated Conv + BN + ReLU,
DownLWGated: Downsample + LWGatedConv,
UpLWGated: Upsample + LWGatedConv

The impact of our filter pruning-based network compression method is summarized in Table 12. The results highlight how pruning leads to significant reductions across all three dimensions: parameter count, the total number of floating-point operations (FLOPs), and inference time (ms).

- The total parameter count is reduced from 12.62M to 3.35M (a reduction of 73.5%).
- The total FLOPs drop from 368.35M to 131.63M (a reduction of 64.3%).
- The total inference time decreases from 3.12 ms to 1.08 ms, achieving a runtime speedup of 2.98 ×.

6.5 Comparison with VR-style Extrapolation Methods

ASW (Asynchronous Spacewarp) [Beeler and Gosalia 2016; Dean et al. 2016] is a technique used in head-mounted displays (HMDs), such as Oculus Quest, to maintain visual smoothness when the VR application does not maintain the required frame rate. It achieves this by extrapolating the previously rendered frame to generate a plausible new frame, leveraging re-projection based on the current head position. In this section, we compare the quality of *PatchEX* against ASW. While most of our test scenes are not compatible with Oculus Quest 2, the Bunker (BK) scene is. As shown in Fig. 12, ASW struggles in complex scenarios involving large disocclusion or nonlinear motion, whereas *PatchEX* significantly outperforms it in terms of image quality and temporal consistency.

6.6 Comparison with Commercial Extrapolation Methods

As mentioned in Section 1, DLSS 3 (Deep Learning Super Sampling) is an advanced frame generation technology developed by NVIDIA to enhance frame rates in real-time graphics applications (released in Oct '22). It utilizes deep learning-based motion vector analysis and optical flow estimation to synthesize intermediate frames, effectively increasing perceived smoothness without requiring additional rendering. By leveraging AI-driven extrapolation, DLSS 3 predicts and generates frames based on historical motion data and scene

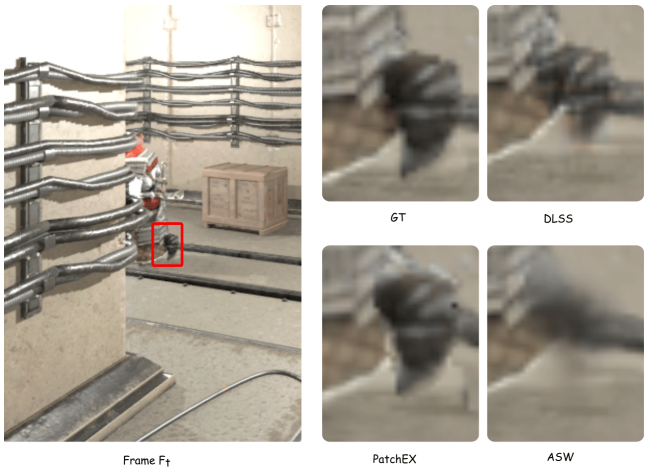


Fig. 12. Quality comparison with the VR-style extrapolation method ASW and NVIDIA DLSS 3

dynamics. Fig. 12 shows a representative result with DLSS 3. We can clearly observe that *PatchEX* is quite close to the ground truth (GT), and the discrepancy of the image generated by DLSS 3 is quite high.

6.7 Performance Analysis for High-Resolution Frames

In this area, the standard practice is to perform extrapolation at a lower resolution (360p or 720p), and then use spatial supersampling to increase the resolution of all the frames (original and generated). All the prior works in this area [Guo et al. 2021; Wu et al. 2023a] have done the same. Akin to *PatchEX*, they assume that the spatial supersampling technique is orthogonal. Nevertheless, for gaining valuable insights into the efficiency of our algorithm, let us evaluate its effectiveness when we *directly work* with frames at a full-HD resolution, i.e., 1080p. This is a *thought experiment*. We maintain the same experimental setup and use the same evaluation metrics.

Fig. 13 shows that the extrapolated frames in this setting closely match the ground truth. Table 13 reports the average PSNR, SSIM and LPIPS values across benchmarks for various frame resolutions. In the interest of saving space, we are not showing all the results. However, a comparison with *ExtraSS* that has a built-in spatial supersampler was done. Representative results are shown in Table 13.

Fig. 13. Performance on high-resolution frames

Table 13. Performance of *PatchEX* in terms of average PSNR (dB), SSIM, and LPIPS at various resolution levels.

Scene res.	*PatchEX* PSNR↑	*PatchEX* SSIM↑	*PatchEX* LPIPS↓	ExtraSS PSNR↑	ExtraSS SSIM↑	ExtraSS LPIPS↓
360p	**36.67**	**0.971**	**0.024**	24.70	0.758	0.246
480p	**35.90**	**0.967**	**0.035**	22.98	0.769	0.258
720p	**36.23**	**0.963**	**0.019**	21.88	0.747	0.247
1080p	**36.25**	**0.969**	**0.026**	20.77	0.718	0.218

6.8 Sensitivity Analysis

To quantitatively validate the effectiveness of our foreground background segmentation, we compare our binary segmentation masks with human visual saliency maps generated using MLNet [Cornia et al. 2016], a state-of-the-art model trained to predict perceptual sensitivity or human attention. We use Intersection over Union (IoU) as the evaluation metric to measure how well our segmentation aligns with the predicted salient regions. Specifically, for each frame in our diverse set of scenes, we compute the IoU between our binary segmentation mask and the binarized MLNet saliency map for various thresholds for motion and depth values. Our method achieves a mean IoU of 0.71 when the foreground is defined using a motion threshold of 0.1 (i.e., 10% of the maximum motion value, which is 1 in our HDR linear range [0, 1]) and a scene depth threshold of 0.8 (i.e., selecting pixels with depth values less than 80% of the maximum). These thresholds correspond to our best-performing setting, where foreground regions strongly align with salient areas predicted by MLNet. To further validate the robustness of our segmentation, we evaluate performance across a range of threshold values:

- Motion thresholds: 0.05 (5%), 0.1 (10%), 0.2 (20%)
- Depth thresholds: 0.6 (60%), 0.8 (80%), 0.95 (95%)

These variations help us analyze how sensitive the segmentation quality is to the choice of thresholds. As expected, thresholds that are too lenient (e.g., motion > 0.05 and depth < 0.95) tend to oversegment background regions, lowering IoU scores, while overly strict thresholds (e.g., motion > 0.2 and depth < 0.6) may miss salient parts of the foreground, also leading to suboptimal alignment. These results quantitatively confirm that our segmentation strategy reliably targets perceptually important regions that closely match human visual attention.

6.9 Failure Cases

While *PatchEX* produce plausible results in all the cases, our method has a few limitations. Specifically, we do not incorporate a dedicated algorithm for handling out-of-screen regions, which may impact extrapolation quality near image boundaries, especially during rapid global camera motion. Fig. 14 (top row) illustrates such a case, when the camera moves quickly, noticeable artifacts appear near the image edges. These are the areas that were previously out of the screen and thus lack reference information from earlier frames. Moreover, in scenes with highly complex geometries (e.g., dense foliage), our method can produce visual artifacts at high resolutions. This is shown in the bottom row of Fig. 14, where frames extrapolated at 4K exhibit distortions that were not seen in lower-resolution outputs

such as 1080p (see Fig. 13), indicating that our current network design may not be fully optimized for spatial supersampling.

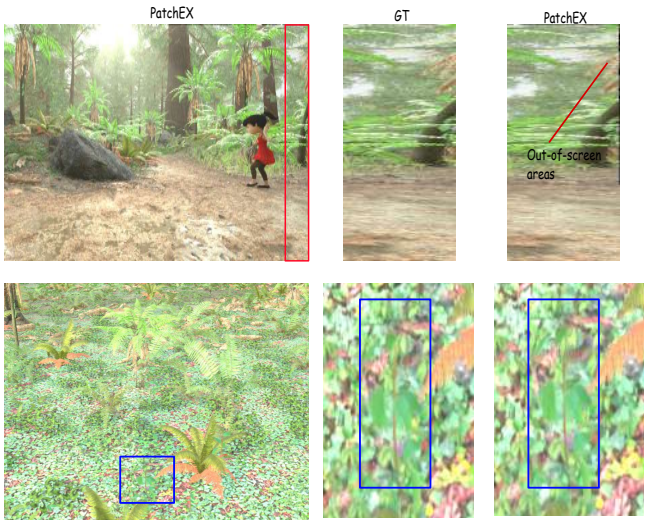


Fig. 14. Failure cases for our method.

7 CONCLUSION AND FUTURE WORK

With high-frequency displays becoming increasingly popular, generating frames for real-time applications at higher rates with superior quality is necessary. Since applications are very demanding in terms of processing power, even most GPUs cannot provide a consistently high frame rate at an HD or 4K resolution. This work illustrates one such method, *PatchEX*, that strives to provide the quality of interpolation with the latency of extrapolation. We proposed a perceptually guided neural network that adaptively processes foreground and background regions using deformable convolutions. Furthermore, to reduce the model size and inference time, we employed a novel wavelet transform-based filter pruning approach that prunes redundant filters. We achieved an improvement of 48.46% in quality (PSNR) and 2.6× better runtime as compared to the nearest competing work ExtraSS. As future work, we plan to develop a unified framework that jointly addresses temporal extrapolation and spatial supersampling, aiming to enhance both temporal consistency and spatial fidelity in high-resolution frame generation.

ACKNOWLEDGMENTS

This research was supported by the Ministry of Electronics and Information Technology (MeitY), Government of India (Grant #3095895).

REFERENCES

2023. Gaming Monitors Market Trend, Share, Growth, Size and Forecast 2030. https://www.zionmarketresearch.com/report/gaming-monitors-market-size. [Online; accessed 2024-05-15].

2024. Mixamo. https://www.mixamo.com/. [Online; accessed 2024-04-10].

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 2010 Talks*. ACM, LA, USA, 1–1.

Anonymous Authors. 2024. Video Results. https://drive.google.com/drive/folders/1hfo7vzAFXd3SzJ0MbtLXpD_TnvTTsc9K. [Online; accessed 2024-06-12].

Dean Beeler and Anuj Gosalia. 2016. Asynchronous Timewarp on Oculus Rift. https://developers.meta.com/horizon/blog/asynchronous-timewarp-on-oculus-rift/. [Online; accessed 2025-02-24].

Yash Belhe, Michaël Gharbi, Matthew Fisher, Iliyan Georgiev, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Discontinuity-Aware 2D Neural Fields. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–11.

Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. 2000. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 417–424.

Mårten Björkman and Jan-Olof Eklundh. 2005. Foveated figure-ground segmentation and its role in recognition. In *2005 16th British Machine Vision Conference, BMVC 2005, Oxford, United Kingdom, 5 September 2005 through 8 September 2005*. British Machine Vision Association, BMVA, Oxford, United Kingdom, 819–828.

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.

Andrew Burnes and Henry C Lin. 2023. NVIDIA ADA Science. https://images.nvidia.com/aem-dam/Solutions/geforce/ada/ada-lovelace-architecture/nvidia-ada-gpu-science.pdf. [Online; accessed 2023-05-09].

Marcella Cornia, Lorenzo Baraldi, Giuseppe Serra, and Rita Cucchiara. 2016. A deep multi-level network for saliency prediction. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 3488–3493.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. 2017. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*. 764–773.

Beeler Dean, Hutchins Ed, and Pedriana Paul. 2016. Asynchronous Spacewarp. https://developers.meta.com/horizon/blog/asynchronous-spacewarp/. [Online; accessed 2025-02-24].

Gyorgy Denes, Akshay Jindal, Aliaksei Mikhailiuk, and Rafał K Mantiuk. 2020. A perceptual model of motion quality for rendering with adaptive refresh-rate and resolution. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 133–1.

Akanksha Dixit and Smruti R Sarangi. 2024. PredATW: Predicting the Asynchronous Time Warp Latency For VR Systems. *ACM Transactions on Embedded Computing Systems* 23, 5 (2024), 1–37.

Suzette Fernandes and Monica S Castelhano. 2021. The foreground bias: Initial scene representations across the depth plane. *Psychological Science* 32, 6 (2021), 890–902.

Epic Games. 2023a. Marketplace. https://www.unrealengine.com/marketplace/en-US/store. [Online; accessed 2023-05-23].

Epic Games. 2023b. Unreal Engine 5. https://www.unrealengine.com/en-US/unreal-engine-5. [Online; accessed 2024-04-10].

Felix Gembler, Piotr Stawicki, Aya Rezeika, Abdul Saboor, Mihaly Benda, and Ivan Volosyak. 2018. Effects of monitor refresh rates on c-VEP BCIs. In *Symbiotic Interaction: 6th International Workshop, Symbiotic 2017, Eindhoven, The Netherlands, December 18–19, 2017, Revised Selected Papers 6*. Springer, 53–62.

Christine Guillemot and Olivier Le Meur. 2013. Image inpainting: Overview and recent advances. *IEEE signal processing magazine* 31, 1 (2013), 127–144.

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 Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

Chengcheng Han, Guanghua Xu, Xiaowei Zheng, Peiyuan Tian, Kai Zhang, Wenqiang Yan, Yaguang Jia, and Xiaobi Chen. 2022. Assessing the effect of the refresh rate of a device on various motion stimulation frequencies based on steady-state motion visual evoked potentials. *Frontiers in Neuroscience* 15 (2022), 757679.

Ruian He, Shili Zhou, Yuqi Sun, Ri Cheng, Weimin Tan, and Bo Yan. 2024. Low-latency Space-time Supersampling for Real-time Rendering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. AAAI Press, Washington, DC, USA, 2103–2111.

Robert Herzog, Elmar Eisemann, Karol Myszkowski, and H-P Seidel. 2010. Spatio-temporal upsampling on the GPU. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, Washington DC, USA, 91–98.

Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE, 2366–2369.

Juho Pekka Huhti. 2019. The Effect of High Monitor Refresh Rate on Game Experience. (2019).

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer, Amsterdam, The Netherlands, 694–711.

Devi Klein, Josef Spjut, Ben Boudaoud, and Joohwan Kim. 2024. Variable Frame Timing Affects Perception of Smoothness in First-Person Gaming. In *2024 IEEE Conference on Games (CoG)*. IEEE, 1–8.

BiBi LED. 2025. Introduction. https://www.bibiled.com/how-does-a-high-refresh-rate-led-display-help-achieve-zero-delay-in-financial-information/. [Online; accessed 2025-04-12].

Sungkil Lee, Younguk Kim, and Elmar Eisemann. 2018. Iterative depth warping. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 1–13.

Zhan Li, Carl S Marshall, Deepak S Vembar, and Feng Liu. 2022. Future Frame Synthesis for Fast Monte Carlo Rendering. In *Graphics Interface 2022*. CHCCS, Montreal, Quebec, Canada, 74–83.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1529–1538.

Shuying Liu and Weihong Deng. 2015. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. IEEE, Kuala Lumpur, Malaysia, 730–734. https://doi.org/10.1109/ACPR.2015.7486599

Shengmei Liu, Atsuo Kuwahara, James J Scovell, and Mark Claypool. 2023. The effects of frame rate variation on game player quality of experience. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, Hamburg, Germany, 1–10.

Yajun Liu, Kefeng Fan, and Wenju Zhou. 2024. FPWT: Filter pruning via wavelet transform for CNNs. *Neural Networks* 179 (2024), 106577. https://doi.org/10.1016/j.neunet.2024.106577

Joerg H Mueller, Philip Voglreiter, Mark Dokter, Thomas Neff, Mina Makar, Markus Steinberger, and Dieter Schmalstieg. 2018. Shading atlas streaming. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–16.

Koshiro Murakami, Kazuya Miyashita, and Hideo Miyachi. 2021. A Study on the Relationship Between Refresh-Rate of Display and Reaction Time of eSports. In *Advances on P2P, Parallel, Grid, Cloud and Internet Computing: Proceedings of the 15th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2020) 15*. Springer, 339–347.

Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. 2007. Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, Vol. 41. ACM, California, USA, 61–62.

Netflix. 2016. Toward A practical perceptual video quality metric. https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652. [Online; accessed 2025-06-11].

Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for low-latency direct-touch input. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, New York, USA, 453–464.

Simon Niklaus and Feng Liu. 2020. Softmax splatting for video frame interpolation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5437–5446.

T. Ojala, M. Pietikainen, and T. Maenpaa. 2002. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 7 (2002), 971–987. https://doi.org/10.1109/TPAMI.2002.1017623

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, Vol. 32. NeuRIPS, Vancouver, Canada.

Cai Ran, Xinfu Li, and Fang Yang. 2023. Multi-Step Structure Image Inpainting Model with Attention Mechanism. *Sensors* 23, 4 (2023). https://doi.org/10.3390/s23042316

Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Babak Naderi, Carsten Griwodz, and Sebastian Möller. 2020. A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience. In *Proceedings of the 11th ACM Multimedia Systems Conference*. ACM, Virtual, 15–25.

Joi Shimizu, Heming Sun, and Jiro Katto. 2022. Forward and Backward Warping for Optical Flow-Based Frame Interpolation. In *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, IEEE, Jeju Island, Korea, 082–086.

Maria Shugrina, Ziheng Liang, Amlan Kar, Jiaman Li, Angad Singh, Karan Singh, and Sanja Fidler. 2019. Creative flow+ dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, California, USA, 5384–5393.

Gerrit A Slavenburg, Marcel Janssens, Luis Lucas, Robert Jan Schutten, and Tom Verbeure. 2020. 46-1: Invited Paper: Variable Refresh Rate Displays. In *SID Symposium Digest of Technical Papers*, Vol. 51. Wiley Online Library, 669–672.

Philipp Wagner. 2011. Local Binary Patterns. https://www.bytefish.de/blog/local_binary_patterns.html. [Online; accessed 2024-05-24].

Martin Weier, Michael Stengel, Thorsten Roth, Piotr Didyk, Elmar Eisemann, Martin Eisemann, Steve Grogorick, André Hinkenjann, Ernst Kruijff, Marcus Magnor, et al. 2017. Perception-driven accelerated rendering. *Computer Graphics Forum* 36, 2 (2017), 611–643.

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 *SIGGRAPH Asia 2023 Conference Papers*. ACM, LA, USA, 1–11.

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 *SIGGRAPH Asia 2023 Conference Papers*. 1–11.

Xiaokun Xu and Mark Claypool. 2024. User Study-based Models of Game Player Quality of Experience with Frame Display Time Variation. In *Proceedings of the 15th ACM Multimedia Systems Conference*. ACM, Bari, Italy, 210–220.

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*. 1–11.

Zili Yi, Qiang Tang, Shekoofeh Azizi, Daesik Jang, and Zhan Xu. 2020. Contextual residual aggregation for ultra high-resolution image inpainting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7508–7517.

Zheng Zeng, Shiqiu Liu, Jinglei Yang, Lu Wang, and Ling-Qi Yan. 2021. Temporally Reliable Motion Vectors for Real-time Ray Tracing. *Computer Graphics Forum* 40 (2021), 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.

YanCi Zhang, XueHui Liu, and EnHua Wu. 2003. Accelerated backward warping. *Journal of Computer Science and Technology* 18, 1 (2003), 48–55.

A APPENDIX

A.1 Performance of Various Warping Methods

In our scheme, we start with generating a quick *estimate* of the target frame using a warping-based technique. Hence, let us characterize different methods of warping. In Fig. 15, we present the results of various warping techniques: traditional motion vector-based warping, occlusion motion vector-based warping and G-buffer-based warping, in comparison to the ground truth. The results demonstrate that both motion vector-based warping methods exhibit some degree of ghosting, which affects the overall quality of the output. However, in the case of G-buffer-based warping, ghosting is absent but it leads to incorrect shading in the resulting frame. However, as mentioned in Section 2.2, we use occlusion motion vector-based warping in this paper.

A.2 Dataset

To demonstrate the effectiveness of *PatchEX*, we have created dedicated scenes featuring both camera motion and complex object movements. Specifically, we introduce the following variations:

- **Camera Motion:** We incorporate two types of camera motion: (i) the camera tracking moving objects and (ii) the camera following predefined linear trajectories (left, right, zoom in, zoom out, tilt).
- **Complex Object Motion:** Our dataset includes objects performing diverse, intricate motions such as hip-hop dancing, push-ups, and ascending stairs.
- **Variable Motion Speeds:** To capture different motion dynamics, we systematically vary the speed of movement across frames.
- **Multiple Moving Objects:** We include scenes with multiple independent objects located at varying scene depths, moving in different directions and at different speeds. These interactions create high motion complexity and diverse disocclusion regions as objects pass each other.

This ensures a more comprehensive evaluation of *PatchEX* in various real-world motion scenarios.

A.3 LBP Computation

We compute LBP input on-the-fly during runtime using OpenCV integrated into Unreal Engine. We use the basic 3×3 LBP operator, where each pixel compares its intensity with 8 surrounding neighbors at a radius of 1, also known as LBP(8,1) [Ojala et al. 2002]. The RGB frame is first converted to grayscale, and then the LBP is computed using a lightweight custom implementation based on OpenCV's primitives [Wagner 2011]. This computation runs in parallel with the warping process using Unreal Engine's multi-threading capabilities to minimize overhead. Once computed, the LBP map is passed as an input channel to the neural network, alongside other inputs.

A.4 Variation in the Frame Rate

As highlighted in Section 1, visual artifacts such as screen tearing and judder can occur even with a high refresh rate display. This is often due to the irregular delivery of frames from the GPU. This section delves deeper into the variability of frame rates in real-world applications. We conducted extensive experiments, measuring the total rendering time for each frame in a scene with a single dynamic object. By plotting these values over time (see Fig. 16), we observe significant fluctuations in the FPS (frames per second) with some frames taking considerably longer to render. The average FPS is almost 29, while the standard deviation is 6.6, indicating that the FPS values show considerable variability or spread around the mean. This inconsistency leads to noticeable flickering, distracting viewers and degrading image quality. Our findings underscore the necessity of temporal supersampling to stabilize the frame rate, aligning it more closely with the display's refresh rate, and thus ensuring high-quality, flicker-free images.

When a frame is being rendered, it passes through a series of steps that form the rendering pipeline. To better understand the reasons behind the high rendering time and variability, we conducted a detailed analysis of the pipeline. We identified the top ten high-latency steps that cause delays and plotted them in Fig. 17. The plot shows that the ShadowDepths, BasePass, PrePass and ShadowProjections steps are the most time-consuming ones. These steps involve complex calculations and require significant computing resources, which can result in high rendering times.

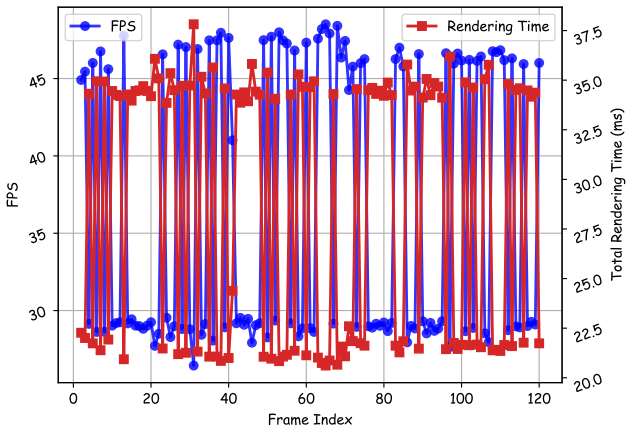


Fig. 16. Variation in the total rendering time

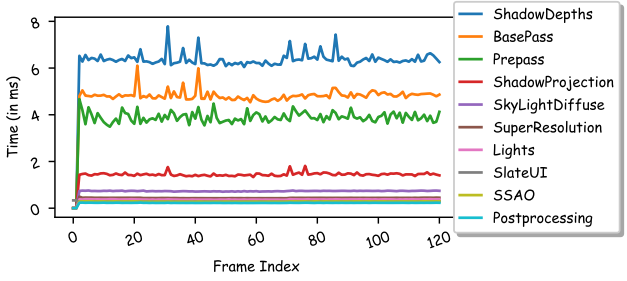


Fig. 17. Top 10 high-latency steps in the rendering process

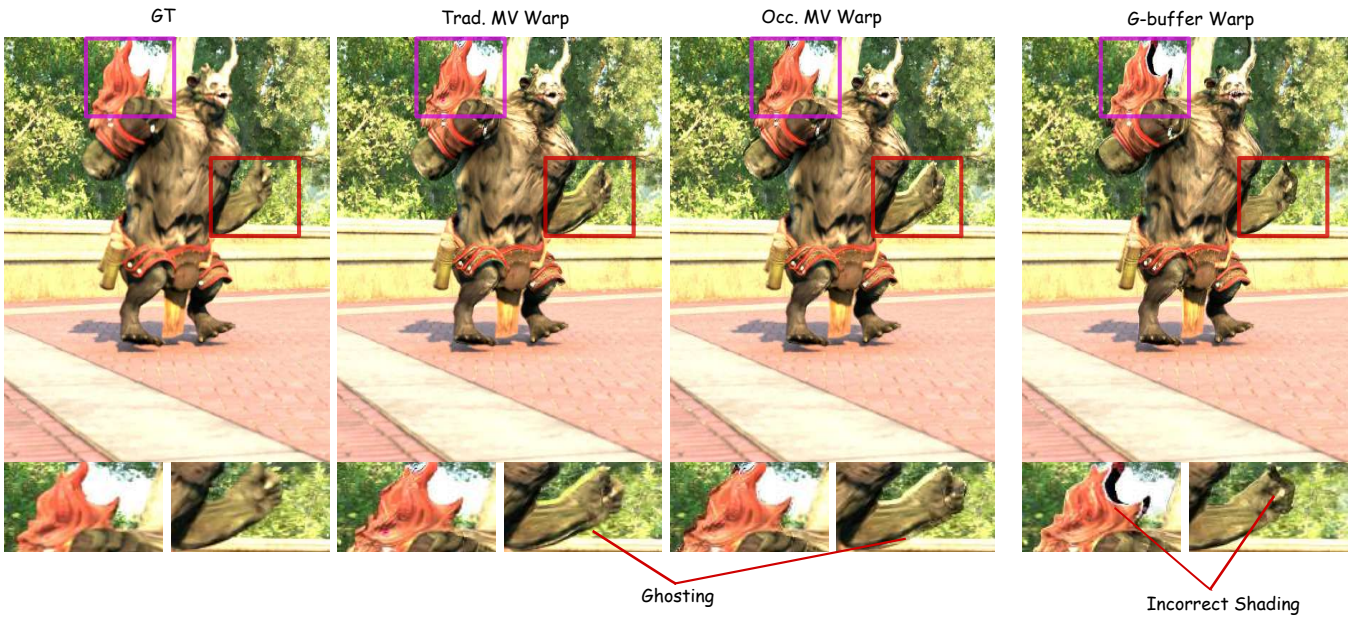


Fig. 15. Comparison of various warping techniques

Insight: For a real-world application, we achieve an average FPS of 29 with a standard deviation of 6.6. This is notably lower than the typical 60 Hz refresh rate of most monitors, potentially leading to visible stuttering or motion instability. This highlights the necessity of using frame generation methods to ensure a smoother and more stable visual experience.

A.5 Deformable Convolution

Recall that a standard convolution operation applies a fixed kernel across an input feature map, computing the output at location $\mathbf{p}_0$ as:

$$Y(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} W(\mathbf{p}_n) \cdot X(\mathbf{p}_0 + \mathbf{p}_n), \tag{11}$$

where:

- $\mathcal{R}$ is the fixed receptive field of size $K \times K$.
- $W(\mathbf{p}_n)$ represents the convolutional weights.
- $X(\mathbf{p}_0 + \mathbf{p}_n)$ is the sampled input feature at position $\mathbf{p}_0 + \mathbf{p}_n$.

However, standard convolution is limited by its fixed sampling locations, making it less effective for capturing deformations in spatial structures. Deformable convolution enhances standard convolution by introducing **learnable offsets** $\Delta\mathbf{p}_n$, allowing dynamic sampling at adaptive locations:

$$Y(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} W(\mathbf{p}_n) \cdot X(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n). \tag{12}$$

Here, $\Delta\mathbf{p}_n$ represents the offset at each sampling position, learned via an additional **offset prediction** network:

$$\Delta\mathbf{p} = f_\theta(X), \tag{13}$$

where f_θ is a convolutional layer that predicts offsets from input features. Also, we apply deformation based on the motion vector and adjust the offset scaling dynamically based on motion magnitude. Specifically: Large motion → Larger receptive field (increased offset). Small motion → Smaller receptive field (reduced offset).

A.6 Discussion with the Concurrent Work

Concurrent work, Mob-FGSR [Yang et al. 2024], also investigates frame generation, with a focus on mobile platforms. They propose a lightweight motion vector (MV) reconstruction method to perform extrapolation; however, this approach can lead to ghosting artifacts in disoccluded regions. Furthermore, due to computational constraints, Mob-FGSR does not account for shading variations, which causes artifacts in visually critical elements such as shadows, reflections, and transparent objects. In contrast, our method uses an occlusion-aware MV-based warping strategy combined with an inpainting network to address these challenges. While Mob-FGSR prioritizes efficiency at the cost of visual quality, we achieve both high quality and fast inference through a filter pruning-based compression technique.

Received 6 July 2024; revised 25 April 2025; accepted 24 July 2025