# ExtraNet: Real-time Extrapolated Rendering for Low-latency Temporal Supersampling

JIE GUO, XIHAO FU, LIQIANG LIN, HENGJUN MA, and YANWEN GUO*, State Key Lab for Novel Software Technology, Nanjing University, China
SHIQIU LIU, NVIDIA Corporation, United States of America
LING-QI YAN, University of California, Santa Barbara, United States of America

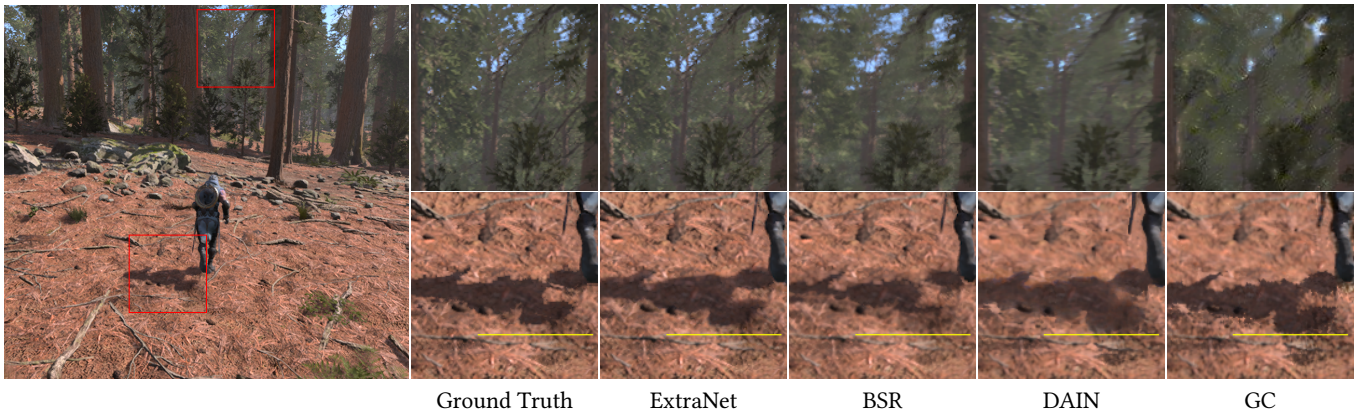



Fig. 1. The proposed ExtraNet is designed to extrapolate a novel frame from historical frames in an animated sequence. It significantly outperforms previous frame interpolation (i.e., BSR [Yang et al. 2011] and DAIN [Bao et al. 2019]) or extrapolation (i.e., GC [Yu et al. 2019]) methods both qualitatively and quantitatively. The predicting time is much faster than actual rendering, which allows low-latency temporal supersampling. Yellow lines in the closeups highlight that our method succeeds in capturing the plausible shadow movement which is hard to be handled by existing methods.

Both the frame rate and the latency are crucial to the performance of real-time rendering applications such as video games. Spatial supersampling methods, such as the Deep Learning SuperSampling (DLSS), have been proven successful at decreasing the rendering time of each frame by rendering at a lower resolution. But temporal supersampling methods that directly aim at producing more frames on the fly are still not practically available. This is mainly due to both its own computational cost and the latency introduced by interpolating frames from the future. In this paper, we present ExtraNet, an efficient neural network that predicts accurate shading results on an extrapolated frame, to minimize both the performance overhead and the latency. With the help of the rendered auxiliary geometry buffers of the extrapolated frame, and the temporally reliable motion vectors, we train our ExtraNet to perform two tasks simultaneously: irradiance in-painting for regions that cannot find historical correspondences, and accurate ghosting-free shading prediction for regions where temporal information is available. We present a robust hole-marking strategy to automate the classification of these tasks, as well as the data generation from a series of high-quality production-ready scenes. Finally, we use lightweight gated convolutions to enable fast inference. As a result, our ExtraNet is able to produce plausibly extrapolated frames without easily noticeable artifacts, delivering a 1.5× to near 2× increase in frame rates with minimized latency in practice.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Image manipulation**.

Additional Key Words and Phrases: extrapolation, low-latency, temporal, supersampling

**ACM Reference Format:**
Jie Guo, Xihao Fu, Liqiang Lin, Hengjun Ma, Yanwen Guo, Shiqiu Liu, and Ling-Qi Yan. 2021. ExtraNet: Real-time Extrapolated Rendering for Low-latency Temporal Supersampling. *ACM Trans. Graph.* 40, 6, Article 1 (December 2021), 16 pages. https://doi.org/10.1145/3478513.3480531

## 1 INTRODUCTION

The compute workload for modern real-time rendering applications have been steeply increasing over the last few years. Users of real-time rendering application always have to make compromises and make a balanced trade-off among image quality, frame rate and resolution. Especially with the recent rapid adoption of real-time ray tracing in the gaming industry, even high end consumer GPUs fails to deliver a consistent 60 FPS experience at common resolutions such as 1440P or 4K.

*Corresponding author

Authors' addresses: Jie Guo, guojie@nju.edu.cn; Xihao Fu, fuxihao66@gmail.com; Liqiang Lin, linlq2017@gmail.com; Hengjun Ma, 171860627@smail.nju.edu.cn; Yanwen Guo, ywguo@nju.edu.cn, State Key Lab for Novel Software Technology, Nanjing University, China; Shiqiu Liu, NVIDIA Corporation, United States of America, edliu@nvidia.com; Ling-Qi Yan, University of California, Santa Barbara, United States of America, lingqi@cs.ucsb.edu.


0730-0301/2021/12-ART1 $15.00
https://doi.org/10.1145/3478513.3480531

Many techniques have been developed in order to increase the rendering performance. Among these techniques, those leveraging the temporal coherency between frames have been widely adopted [Scherzer et al. 2012], including TAA (Temporal Anti-Aliasing) [Xiao et al. 2018; Yang et al. 2020, 2009], TAAU (Temporal Anti-Aliasing Upsample) [Epic Games 2018], TRM (Temporal Resolution Multiplexing) [Denes et al. 2019], DLSS (Deep Learning Super Sampling) [Liu 2020], and various denoising algorithms developed specifically for ray tracing [Bako et al. 2017; Chaitanya et al. 2017; Guo et al. 2019; Schied et al. 2017; Sen et al. 2015; Vogels et al. 2018]. These techniques all reduce the sampling rate at each individual frame, and rely on temporal reprojections to combine samples taken over multiple frames to reconstruct a high quality image.

Although pervasively applying the temporal information, these methods are essentially still performing spatial upsampling / reconstruction, with a virtually higher sampling rate acquired temporally to speed up the generation of the current frame of interest. In this paper, we present an orthogonal approach to the temporal reconstruction family, aiming at *directly increasing the frame rate / temporal sampling rate* of rendered sequences.

While prominent advances have been made to increase frame rates directly, existing approaches are not suitable for our purpose. The most critical factor is the latency, since most previous attempts refer to *frame interpolation* [Mark et al. 1997; Yang et al. 2011]. That is, only when the next frame has been fully rendered, their algorithms can start to work to interpolate one or more frames in between.

Another important factor to consider is the performance. This is intuitive: if generating a new frame is already more expensive than a full rendering of it, it does not make sense to use temporal supersampling with a higher cost but a potentially lower quality, since after all, what it does is to hallucinate a frame. Unfortunately, even the slowest real-time rendering application is able to generate a frame in tens of milliseconds. This immediately rules out a lot of work that produces excellent quality but runs very slow [Bao et al. 2019; Huang et al. 2020; Niklaus and Liu 2020; Xiang et al. 2020].

The third factor is the situation they apply. We focus on rendered scenes, which means that there are more information to use, especially the G-buffers that do not exist in video interpolation / extrapolation, such as Oculus' ASW (Asynchronous Space Warp) technology [Oculus 2016]. In general, auxiliary geometry buffers (or G-buffers, such as per pixel normals and depths) are relatively cheap to acquire (usually within several milliseconds per frame) as compared to heavy shading tasks and potential post processing. This property brings us a unique advantage. That is, we are able to generate the G-buffers for the extrapolated frame, then use them to guide the extrapolation of shading on these G-buffers.

Based on the above analysis, we present ExtraNet, a neural network that extrapolates the shading in image space based on previously rendered frames. To achieve optimal image quality in the extrapolated frames, our method takes advantage of the G-buffers from the extrapolated frame rendered by the Unreal Engine 4, including depth, normals, albedos, material definitions.

The G-buffers also directly enables us to use the temporally reliable motion vectors [Yang et al. 2020; Zeng et al. 2021], which help us to find accurate per-pixel temporal correspondences, and to classify our tasks into two: irradiance in-painting for regions that cannot find historical correspondences, and accurate ghosting-free shading prediction for regions where temporal information is available. We train our ExtraNet to perform two tasks simultaneously. Moreover, we present a robust hole-marking strategy to automate the data generation from a series of high-quality production-ready scenes. Finally, we use light-weight gated convolutions to enable fast inference. Since additional computational costs for generating G-buffers are required and complex neural networks are involved, our method runs slightly slower than some traditional methods relying on fast image warping and reprojection. Nevertheless, the image quality of extrapolated frames has been significantly improved, as we show in Sec. 6. We believe a specialized and efficient CUDA implementation that fully utilizes tensor cores will further accelerate the pipeline.

With our technique, the rendering engine could interleave regular rendering and shading extrapolation when rendering a sequence, which would nearly double the frame rate, but without easily noticeable artifacts. Moreover, our approach does not introduce any additional latency as we don't require the future frame to finish rendering.

To summarize, our contributions in this work include:

(1) A new temporally interleaved real-time rendering and extrapolation architecture that can nearly double the frame rate without introducing additional latency.
(2) A novel neural network architecture that leverages G-buffers and temporal motion (computed both from consecutive frames and temporally reliable motion vectors) to perform the extrapolation as a shading prediction task.
(3) A lightweight design that incurs low performance overhead (around 8 milliseconds per frame at 720P) but results in high quality and smoothly extrapolated frames.

## 2 RELATED WORK AND BACKGROUND

There are several classes of previous work that are related to our work. In particular, temporal reconstruction for real-time rendering (including temporal antiliasing and temporal denoising), texture in-painting, image warping, and interpolation based frame rate upsampling for videos / rendering.

### 2.1 Temporal Reconstruction

To achieve high quality frame extrapolation, we heavily rely on (backward) temporal motion vectors, which is also used by almost all the common temporal reconstruction methods [Scherzer et al. 2012]. Temporal reconstruction methods reduce the samples taken at each individual frame, and reproject samples over the course of multiple frames using backward motion vectors to reconstruct a high quality image. Schied et al. [2017] estimate the spatiotemporal variance to guide the filtering parameters per pixel. Chaitanya et al. [2017] propose a recurrent autoencoder to automatically accumulate historical information. Zimmer et al. [2015] compute motion vectors of specular paths with a general decomposition framework and a temporal extension of manifold exploration. Mara et al. [2017] improve the motion vectors for specular reflections, by tracking the movement of the specular reflected virtual images instead of the "mirrors". Vogels et al. [2018] use consecutive frames to guide the

denoising of the frame of interest to reduce flickering in production scenes. Recent work by Zeng et al. [2021] introduces temporally reliable motion vectors that suppress ghosting artifacts from shadows, glossy reflections and disocclusions. The success and popularity of the motion vectors also indicate that the change of shading in a fixed pixel should not be simply treated as linear over time [Mueller et al. 2021] for high quality temporal reconstruction.

We also leverage temporal motion vectors to reproject shading of the previous rendered frames to the extrapolated frames. However, unlike temporal reconstruction, which still take a small amount of shading samples at the frame being reconstructed, we do not have any samples in the extrapolated frames. Therefore, we are essentially reconstructing / "denoising" the full shading of an extrapolated frame using 0 shading samples per pixel (except for the G-buffers). This analogy connects our extrapolation with existing temporal reconstruction work, and also demonstrates the significant difficulty that our extrapolation task faces.

### 2.2 Texture In-painting

One caveat of using backward motion vectors to reproject shading from previous frame is that disoccluded regions will not contain any valid information in the previous frame, resulting in "holes" in the reprojected image. Filling the holes in an plausible way is a critical element of our technique. Previously, numerous research has been done in the area of texture in-painting using deep learning. Some methods use convolution variants, such as partial convolution [Liu et al. 2018], gated convolution [Yi et al. 2020; Yu et al. 2019] and masked convolution [Santos et al. 2020], to extract reliable features from valid regions and restore the image with these features. Currently, these methods have been favored by many applications due to their simplicity and high efficiency. Other in-painting methods resort to a two-stage prediction strategy to separately predict missing textures in a step-by-step manner [Liu et al. 2020; Ren et al. 2019; Xiong et al. 2019; Yu et al. 2018].

However, these techniques are not designed specifically for real-time rendering. They perform the in-painting task completely on single images without temporal information and G-buffers, which can often lead to unbounded failures when they fail to generalize to new content.

### 2.3 Image Warping

Image warping has a long history in both computer graphics and vision. Here, we focus on efficient techniques for real-time rendering which are closely related to our work. Typically, image warping is realized by forward scattering or backward gathering, according to their data access patterns. Mark et al. [1997] leverage a forward mapped image warping algorithm to scatter pixels to polygonal meshes, achieving post-rendering 3D warping. Didyk et al. [2010a] use accurate motion vectors to compute a forward image warp defined by a coarse grid representation. The quality of warped images can be further improved by using adaptive grid refinement [Didyk et al. 2010b; Schollmeyer et al. 2017] or geometry proxies [Reinert et al. 2016]. Compared with forward mapping, backward mapping is more appealing in real-time rendering due to its high efficiency. Andreev [2010] proposes to use half motion vector of next frame to warp previous frame. Yang et al. [2011] interpolate a pair of consecutive rendered frames with bidirectional reprojection. Bowles et al. [2012] establish a general framework for backward image warping using fixed point iteration. Although visually plausible contents can be recovered at hole regions after image warping, shadow and highlight movements are almost untouched in these traditional methods, especially those used in the context of frame extrapolation.

### 2.4 Video and Rendering Interpolation

Interpolation based frame rate upsampling is widely used for temporally upsampling videos [Baker et al. 2007; Shechtman et al. 2010]. These techniques are often based on computing the forward and backward optical flow [Fortun et al. 2015] of the last couple of frames in the video sequences. Then the optical flow can be used to reproject the future and past frames to get an estimate of the intermediate frames. However, optical flow is an image based algorithm that often produces inaccurate motion, which will lead to artifacts in the interpolation results. Our method instead uses accurate motion vectors available from the rendering engine, so the reprojection quality is a lot better, which is verified in several existing work [Didyk et al. 2010a,b; Leimkühler et al. 2017; Mark et al. 1997; Yang et al. 2011]. These methods usually rely on image warping techniques to reuse shading information across frames.

Although recent deep learning based video interpolation methods such as [Bao et al. 2019, 2021; Huang et al. 2020; Jiang et al. 2018; Niklaus and Liu 2020] have made a great progress on enhancing the quality of flow prediction and warping occlusion, these methods are still time-consuming, thus are not suitable for real-time rendering.

All these interpolation methods, however, have to require a future frame to be rendered first, therefore delaying the display of every rendered frame, introducing additional input latency. This can be a very undesirable trait for video game players. Our technique, on the other hand, incurs no additional latency.

## 3 MOTIVATION AND CHALLENGES

Before we describe our method, let us briefly analyze our temporal supersampling problem. Specifically, we would like to understand our advantages and challenges.

### 3.1 Latency

In almost all rendering engines, rendered frames are first saved in render targets (frame buffers) before actual display (scheduled by drivers). A synchronization mechanism (e.g., V-sync or G-Sync) is usually adopted to ensure constant frame rates even if the frames are generated with varying time. This is particularly important for frame interpolation / extrapolation since interpolated / extrapolated frames often consume much less time than conventional rendering, as demonstrated in Fig. 2. The gap between frame generation and display incurs latency in rendering engines.

There are various definitions and types of latency from the industry. Here we provide a definition of it – the delay between the moment a frame has been generated and the moment it starts to be presented to users by the GPU. This definition completely rules out the latency from network conditions, refresh rates of monitors, etc.,

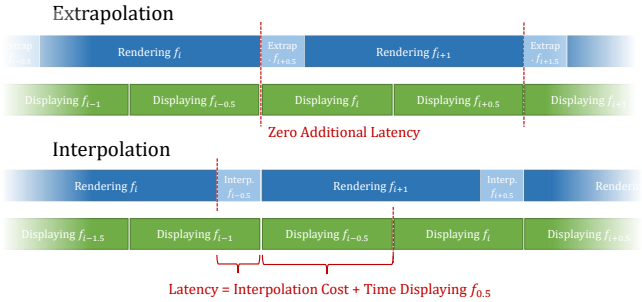



Fig. 2. An illustration of different latency induced by extrapolation and interpolation, respectively.

and is only related to rendering and the scheduling of display buffers. We make it even simpler by assuming a perfect frame scheduler. That is, the scheduler can always choose the right buffer to display at the right / perfect time, to guarantee the smoothest streams of frames.

Suppose the actual rendered frames are labeled in integers and the interpolated / extrapolated frames are labeled as $0.5, 1.5, \ldots$, Fig. 2 immediately tells us that using interpolation produces significant latency since the in-between frame can only be generated and displayed after a future frame is processed. In contrast, our extrapolation method can always begin generating the next frame immediately after prior frames are complete, incurring little additional latency. Also note a typical misinterpretation that "since the extrapolation takes time, it introduces latency". The extrapolation does take time but also increases the frame rate. It makes no sense to compare two different sequences with different frame rates (and probably different contents) to find one sequence's latency.

However, we do not overclaim that our extrapolation has zero latency. This is because we did not analyze the outcomes theoretically with non-ideal configurations and potential tricks, such as disabling V-sync, that trade quality for lower latency. Also, in the industry, the rendering time is sometimes considered as part of the "input latency", though it is already considered / penalized when measuring the frame rates. But one point is certain that interpolation is indeed causing a much higher latency than extrapolation, thus is not the desired solution. The importance of low latency has been demonstrated everywhere, from commercial success of high refresh rate monitors and low latency streamed video games, to academic studies [Spjut et al. 2019].

All the above analysis leads to our motivation of using extrapolation to perform temporal supersampling. However, there are significantly more challenges that we need to consider.

## 3.2 Challenges

To derive the shading in the extrapolated frame based on reprojected shading from previous frames and the G-buffers from the current frame without visible artifacts, there are several critical challenges that we need to overcome:

*Disocclusion.* Backward motion vectors may not always exist. The most often encountered case in the disocclusion, i.e., when a region

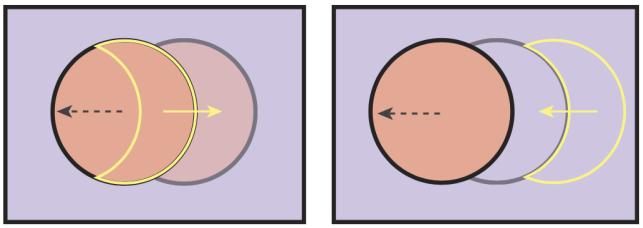


Fig. 3. An illustration of the occlusion motion vectors from Zeng et al. [Zeng et al. 2021]. (Left) Traditional motion vectors propose to use the same pixel values from previous frames, resulting in ghosting artifacts in the disoccluded regions. (Right) occlusion motion vectors alleviates this problem by looking for a nearby similar region in the current frame, then looking for the corresponding region from previous frames. Dashed black vectors indicate object motion, while yellow vectors indicate warping.

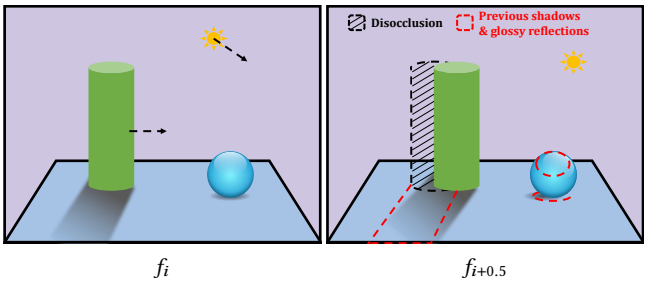



Fig. 4. An illustration of dynamic changes in shading. Shadows and glossy reflections will change drastically when dynamic objects move, as indicated in dashed red regions. Our method not only fills holes in disoccluded regions but also corrects moving shadows and reflections.

is visible in the current frame but the content in the region was occluded previously. In these regions, we are not able to leverage the shadings computed in previous frames since they were occluded. To address this, we leverage an in-painting network to reshade the occluded regions. In order to further improve the reshading quality of our network, we utilize the temporally reliable occlusion motion vector (Fig. 3) proposed in [Zeng et al. 2021] to improve the reprojection quality in disoccluded regions.

*Dynamic changes in shading.* Even in regions that have correspondences from previous frames, it is still not applicable to directly use the shading results from previous frames, because dynamic changes in shading over frames, such as moving shadows, parallax motion from reflections and the colors on rotating objects, may change drastically and are thus unknown in the current frame. This is illustrated in Fig. 4. Simply reusing e.g. the previous one frame's shading will predict exactly the same content, as if the frame rates are not increased at all. *If only disoccluded regions are correctly handled, there will be a high frame rate at these regions, while shadows and glossy reflections still run at a relatively low frame rate*, leading to jerky effects in an animated sequence. This challenge distinguishes our method from pure hole-filling approaches. To address the dynamic shading issue and make correct predictions, we always keep

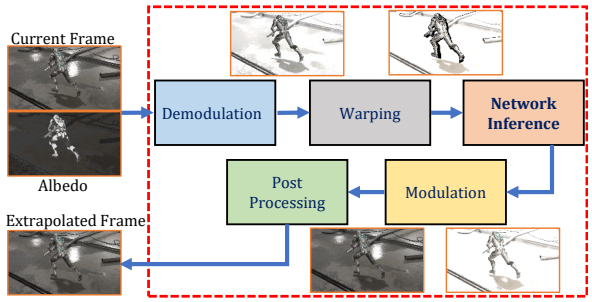



Fig. 5. High-level overview of our pipeline. The current frame is firstly demodulated by its albedo and warped to the space of the next frame to be predicted using motion vectors. After network inference, the generated new frame is modulated by albedo and then apply tone mapping and temporal antialising.

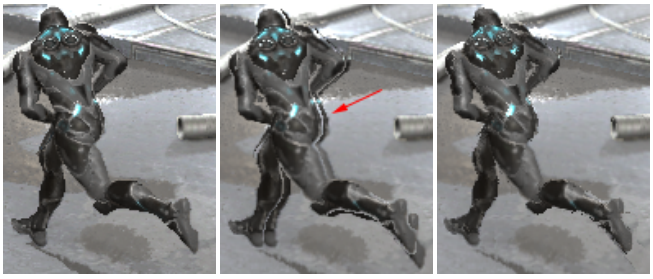



Fig. 6. Illustration of warped images according to different types of motion vectors. Occlusion motion vectors can be used to warp disoccluded pixels which are not correctly handled by traditional motion vectors. Ghosting artifacts are easily observed along the moving character when traditional motion vectors are used.

the most recently rendered 3 frames for our network to learn to extrapolate the changes.

*Other challenges.* Since our method is for real-time rendering applications, one immediate challenge is the extremely low tolerance towards errors and artifacts, because any glaring or flickering will be immediately objectionable to users. Moreover, as pointed out earlier, the inference of the neural network must be (ideally much) faster than the actual rendering process.

## 4 EXTRANET FOR FRAME EXTRAPOLATION

To address the above challenges and enable reliable frame extrapolation for real-time rendering, we design and train a deep neural network, namely ExtraNet, to predict a new frame from some historical frames and the G-buffers of the extrapolated frame.

As analyzed in Sec. 3, our network is designed by taking the following three aspects into consideration. First, considering that disoccluded regions of the extrapolated frame lack shading information but reliable and cheap G-buffers are available, we construct an *In-painting Network* to reshade the disoccluded regions with the help of occlusion motion vectors and those G-buffers. Second, in contrast to the general image in-painting task, regions that are not disoccluded may also contain invalid pixels attributing to dynamic shading changes in the scene. To tackle this issue, we design a *History Encoder*, an additional convolutional neural network, to extract necessary information from historical frames. These information, including potential motions of shadows and reflections, will be concatenated to the In-painting Network, allowing the prediction of proper changes in shadow and reflection areas. From this point, our framework is *far beyond hole filling*. Finally, to meet the requirement of high inference speed which is critical to real-time rendering, we adopt lightweight gated convolutions in our In-painting Network.

### 4.1 Problem Formulation

Now, we formulate our problem in detail. Denoting the current frame rendered by the graphics engine as frame $i$, the goal of our network is to predict the next frame, namely frame $i+0.5$, according to the current frame $i$, as well as two historical frames including frame $i-1$ and frame $i-2$, and the corresponding G-buffers.

Fig. 5 illustrates the high-level overview of our pipeline. We first demodulate (dividing by the albedo to acquire texture-free irradiance) the current frame using the albedo information from the G-buffers, and then warp the demodulated frame using both traditional backward motion vectors and occlusion motion vectors [Zeng et al. 2021].

The warping operation probably incurs holes in the frame, and they are expected to be filled by the In-painting Network. These holes should be properly marked, and the generated masks are also fed into the network. The details of hole marking will be provided in the next section.

After network inference, the generated new frame (frame $i+0.5$) is modulated (multiplying by the albedo to re-acquire textured shading) back by the albedo of frame $i+0.5$. We then apply regular post-processing such as tone mapping and temporal antialiasing to it. Note that the demodulation and modulation are not strictly indispensable in our framework, but we found that using them helps our In-painting Network produce better results in those disoccluded regions.

### 4.2 Motion Vectors and Image Warping

One key step in our pipeline is to warp frames using motion vectors. However, image warping based on traditional motion vectors will easily introduce ghosting artifacts in the disoccluded regions, as illustrated in the middle column of Fig. 6. Though such artifacts can be cleaned up by temporal antialiasing with the help of current frame, it would be impractical in our scenario. Providing incorrectly warped frames to our network will confuse it to generate undesired results. Therefore, besides the frames warped by traditional backward motion vectors, we also leverage occlusion motion vectors proposed by Zeng et al. [2021] to faithfully handle disoccluded regions. Instead of computing a zero motion vector in disoccluded regions as the traditional backward motion vector does, occlusion motion vector computes the motion vector in disoccluded regions as the motion vector of the foreground in the previous frame. Visual comparisons in Fig 6 show that occlusion motion vectors provide better results in the disoccluded regions. Currently, frames warped

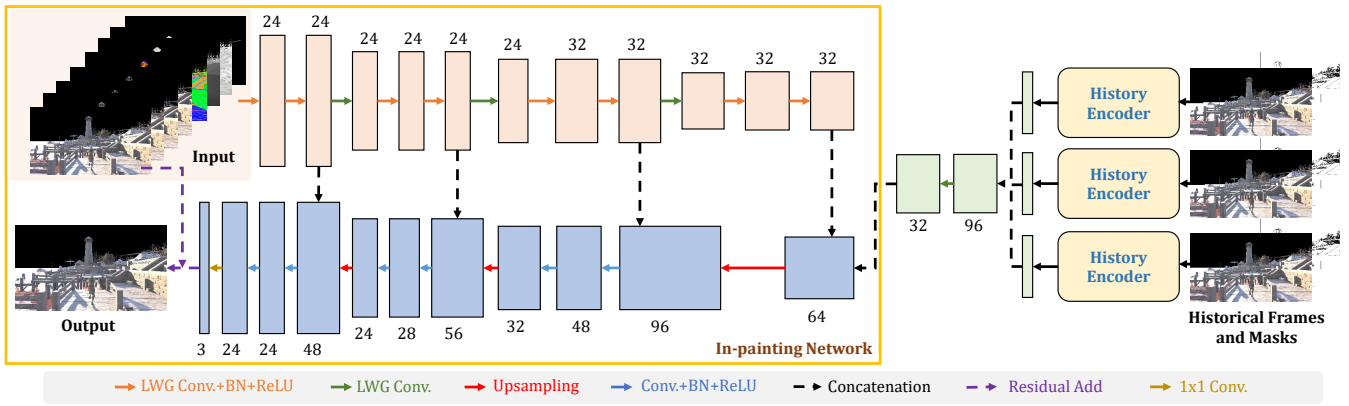



Fig. 7. Network architecture of the proposed ExtraNet. The input to our network comprises two different warped frames, a mask (representing the marked holes) and the corresponding G-buffers. The output is an extrapolated frame with proper changes in occlusions, shadows and reflections. In the In-painting Network, the encoder is stacked by 11 lightweight gated convolution (LWG Conv.) layers, while the decoder is stacked by 7 standard convolution (Conv.) layers. BN represents batch normalization. Three historical frames and their masks pass through History Encoder and the generated feature maps are concatenated to the bottleneck of In-paint Network. Note that all frames shown here are demodulated by albedo.

by traditional motion vectors and occlusion motion vectors are both fed into our ExtraNet.

### 4.3 Network Architecture

The architecture of our proposed ExtraNet is shown in Fig. 7. Our network shares a similar structure with U-NET [Ronneberger et al. 2015] which provides state-of-the-art results in some computer vision tasks. Conventional U-NET, however, cannot be directly used in the in-paining task since the standard convolution operation treats all pixels, including those invalid pixels in the holes, equally. To address this, we adopt gated convolutions [Yu et al. 2019] in our network to provide a learnable feature selection mechanism. Specifically, the output of gated convolutions is computed as

$$\mathbf{M} = \mathrm{Conv}(\mathbf{W}_m, \mathbf{X}), \tag{1}$$

$$\mathbf{F} = \mathrm{Conv}(\mathbf{W}_f, \mathbf{X}), \tag{2}$$

$$\mathbf{O} = \sigma(\mathbf{M}) \odot \mathbf{F}, \tag{3}$$

where $\mathbf{X}$ is an input feature map, and $\mathbf{W}_m$ and $\mathbf{W}_f$ are two trainable filters. $\odot$ denotes element-wise multiplication. With a sigmoid activation $\sigma(\cdot)$, $\mathbf{M}$ is expected to provide dynamical masks for masked feature extraction. This helps to weaken the influence of holes.

Though gated convolutions generate satisfactory results in filling irregular holes in our task, it increases the inference time since the convolutions are doubled due to extra computation of the soft mask $\mathbf{M}$. This is not acceptable in real-time rendering scenarios. In light of this, we resort to a lightweight variant of gated convolution by making $\mathbf{M}$ a single-channel mask, as inspired by [Yi et al. 2020]. Compared to the original gated convolution, the lightweight gated convolution significantly improves the inference performance, while still preserving high image quality. To further increase the inference speed, we opt to not use any gated convolution in the upsampling stage, since we believe that all holes have already been filled in the downsampling stage. Moreover, we use a residual learning strategy in our pipeline by adding the predicted image of our network with the input image warped by traditional backward motion vectors. We observe by experiments that this stabilizes the training process.

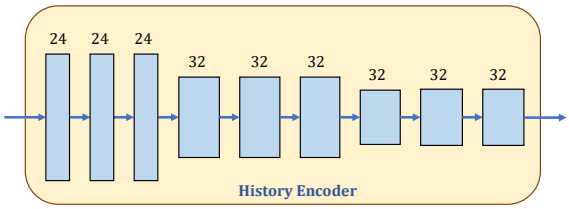



Fig. 8. The detailed architecture of History Encoder. Each layer contains a standard convolution, BN and ReLU activation.

### 4.4 History Encoder

With the above In-painting Network, warped frames with marked holes can be recovered with plausible contents. However, some pixels outside the holes can also be invalid due to the movement of shadows and reflections. Note that these movements are not easily identified by G-buffers. The way we solve this problem is to introduce another network, i.e., History Encoder, to capture these movements by tracking historical frames.

The input to our History Encoder comprises warped images of the current frame $i$ and its two past frames including frame $i-1$ and frame $i-2$. To warp frames $i-1$ and $i-2$, we simply accumulate the motion vectors. Invalid pixels should be marked for these frames, and the corresponding masks are also fed into the History Encoder.

Fig. 8 demonstrates the basic structure of our History Encoder. It is composed of nine $3 \times 3$ convolution layers. The first, fourth and seventh layers are convolution layers with a stride of two, which downsample the input of the convolution layer. This indicates that almost all convolution operations, except the first one, work in a relatively lower resolution. This strategy significantly improves the running performance of History Encoder. To further reduce the

Warped by optical flow GT

Fig. 9. Image warping based on backward optical flow generates improper shadows as highlighted by red lines.

number of parameters in the whole network, History Encoder is shared by different historical frames.

The outputs of History Encoder, which implicitly encode the high-level representations of shadows and shadings in previous frames, are concatenated together and fed into our In-painting Network. Note that concatenation is not the only choice that our In-painting Network can utilize these representations. An alternative representation is optical flow [Fortun et al. 2015] that can be explicitly predicted from several historical frames. Although such representations of movements are popular in many computer vision tasks, such as action recognition [Piergiovanni and Ryoo 2019; Simonyan and Zisserman 2014] and video stabilization [Yu and Ramamoorthi 2020], it is impractical in our application. One main reason is that correct forward optical flow is difficult to acquire in our frame extrapolation pipeline, since only historical frames are available. Consequently, we can only use backward optical flow as an approximation. However, this approximation easily fails when the movement becomes sophisticated, as we demonstrated in Fig. 9. Image warping based on backward optical flow may generate improper shadows that deviate greatly from ground truth.

Another reason for not using optical flow in our pipeline is the lacking of sufficient training examples. Note that the dataset we constructed for training our network has a much smaller size than those used to train optical flow prediction networks in computer vision. A relatively small dataset easily incurs overfitting for the network. In contrast, feature map concatenation that we used in our network can adapt to the dynamic changes in the scene. Moreover, many networks that can predict optical flow run too slow to be used in our real-time pipeline.

### 4.5 Loss Function

We train our whole network, including the In-painting Network and History Encoder, with a joint loss function. The loss function has three components. The first loss penalizes pixel-wise error between the predicted frame $\mathbf{P}$ and ground-truth frame $\mathbf{T}$, which is simply the $L_1$ distance between $\mathbf{P}$ and $\mathbf{T}$:

$$\mathcal{L}_{l_1} = \frac{1}{n}\sum_i |\mathbf{P}_i - \mathbf{T}_i|, \tag{4}$$

where $n$ is the number of pixels in the frame.

Neighbouring frames in a sequence look visually similar. Normally, invalid regions only occupy a very small portion. In-painting results of invalid regions are hardly to be cared for by the traditional $L_1$ loss. To emphasize more on invalid regions, we specifically design another two losses.

The hole-augmented loss $\mathcal{L}_{\text{hole}}$ is designed to penalize more in the hole regions marked beforehand. Let $\mathbf{m}$ be the binary mask fed into the network. The loss is expressed as,

$$\mathcal{L}_{\text{hole}} = \frac{1}{n - \sum_i m_i}\sum_i |\mathbf{P}_i - \mathbf{T}_i| \cdot (1 - m_i) \tag{5}$$

where $m_i$ is the $i$-th element of $\mathbf{m}$.

The shading-augmented loss $\mathcal{L}_{\text{shade}}$ focuses on handling potential shading changes in the predicted frames. These changes may stem from moving shadows due to dynamic lights and specular reflections. Unfortunately, it is not easy to identify these changes in the preprocessing stage. Considering that shadows and reflections tend to generate large pixel variation among neighboring frames, we select $k$ pixels with top $k$ largest errors from the predicted frame and mark these pixels as potential shading change regions. Then, we augment the loss by

$$\mathcal{L}_{\text{shade}} = \frac{1}{k}\sum_{i \in \Phi_{\text{top-k}}} |\mathbf{P}_i - \mathbf{T}_i| \tag{6}$$

where the set $\Phi_{\text{top-k}}$ includes the indices of $k$ pixels with top $k$ largest errors. Even though the top $k$ largest errors may occur in hole regions initially during training, they will move to incorrect shadows and reflections after several iterations when holes have been filled. Currently, we set $k$ to be ten percentage of the total pixel number. The effect of this loss term will be validated in Sec. 6.6.

Our final loss function $\mathcal{L}$ is the summation of these three losses:

$$\mathcal{L} = \mathcal{L}_{l_1} + \lambda_{\text{hole}}\mathcal{L}_{\text{hole}} + \lambda_{\text{shade}}\mathcal{L}_{\text{shade}} \tag{7}$$

where $\lambda_{\text{hole}}$ and $\lambda_{\text{shade}}$ are weights to balance the influence of the losses. In our current implementation, both are set to 1.

### 4.6 Training details

Our ExtraNet is implemented and trained using the PyTorch framework [Paszke et al. 2019]. Mini-batch SGD and Adam optimizer [Kingma and Ba 2014] are used for optimization. Specifically, we set mini-batch size as 8, $\beta_1$ in Adam optimizer as 0.9 and $\beta_2$ as 0.999. Cosine learning rate decay [He et al. 2019] is used after 20 epochs. The initialization of the network follows the default setting in PyTorch. Before feeding images into our network, logarithm transformation $y = \log(1 + x)$ is applied to HDR images to avoid large values.

## 5 DATASET

### 5.1 Scenes and Buffers

To train our network, we have constructed a large-scale dataset using four scenes from Unreal Engine marketplace[1]. These scenes cover a wide range of shading effects, including glossy reflections, soft shadows, multiple lights, and complex occlusions. Each scene contains at least one dynamic object with complex movements. Among these scenes, one (`Western Town`) is only used for testing, while the other three are chosen for both training and testing. Each training sequence contains 900 continuous frames, and each testing sequence contains 300 frames. The splitting of training and testing

[1] https://unrealengine.com/marketplace

Table 1. Statistics of the training dataset and testing dataset used in our pipeline. Here, we list the number of sequences and the number of total frames of each scene.

| Scenes | Training Sequences | Testing Sequences | Training Frames | Testing Frames |
|---|---|---|---|---|
| Medieval Docks | 7 | 5 | 6300 | 1500 |
| Redwood Forest | 7 | 5 | 6300 | 1500 |
| Bunker | 6 | 5 | 5400 | 1500 |
| Western Town | 0 | 5 | 0 | 1500 |

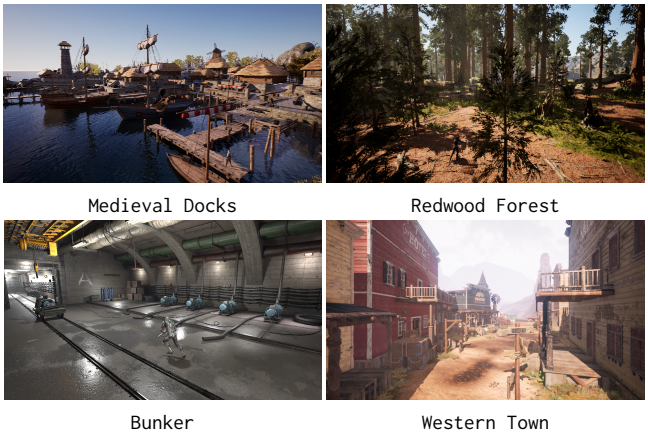



Fig. 10. Example views of four scenes.

sets as well as some statistics are listed in Table 1. Some representative frame examples selected from these scenes are shown in Fig. 10.

Sequential frames are generated on Unreal Engine 4 and dumped into the .exr file format. To warp previous frames to the current frame using motion vectors, we implement a Compute Shader and integrate it into Unreal Engine 4's render pass. Each dumped frame comprises 10 buffers which can be divided into three categories:

(1) Actual shading frame before tone mapping (PreTonemapHDRColor) and albedo (Base color) for demodulation.
(2) G-buffers used in our network, including scene depth ($d$, 1 channel), world normal ($\mathbf{n}_w$, 3 channels), roughness (1 channel), and metallic (1 channel).
(3) Other auxiliary buffers which are necessary for marking holes (invalid pixels) in the warped frame. They are motion vector, world position ($\mathbf{p}_w$), NoV ($\mathbf{n}_w \cdot \mathbf{v}$, the dot product of world normal $\mathbf{n}_w$ and view vector $\mathbf{v}$), and customized stencil ($s$).

Note that in Unreal Engine 4 metallic is used to indicate the metallicity of object. The customized stencil refers to a stencil buffer for masking dynamic objects in the scene. One example of these 10 buffers is shown in Fig. 11.

Our network is currently designed to replace the actual shading pass in the Unreal Engine 4. Therefore, the built-in post-processing passes, such as AA and tone mapping, are still required to generate final sequences. These post-processing passes have little performance cost as compared with G-buffer generation and network inference.

Table 2. Runtime (milliseconds) for generating G-buffers for different scenes. All data are measured on NVIDIA RTX 3090 and frames are rendered at 720P resolution ($1280 \times 720$).

| Scenes | Average | Maximum | Minimum |
|---|---|---|---|
| Medieval Docks | 0.30 | 0.31 | 0.29 |
| Redwood Forest | 2.83 | 4.04 | 2.13 |
| Bunker | 0.14 | 0.18 | 0.14 |
| Western Town | 2.05 | 4.07 | 1.08 |

Note that generating G-buffers usually costs differently among these scenes (see Table 2). The runtime is closely related to the complexity of scene geometries. Nevertheless, rendering G-buffers are much cheaper than actual shading.

## 5.2 Marking Holes

When reusing previous frames to generate a new frame, some pixels in the warped frame will be invalid, due to camera sliding and object moving. These pixels should be marked as holes before feeding into the network. Note that marking pixels as holes does not mean we drop these pixels. Both warped frames and masks will be sent to ExtraNet. Therefore, the network can still extract useful features from the "invalid" pixels.

We mark the pixel as invalid according to the following three conditions. First, for occlusion caused by object moving, we used custom stencil to indicate the sharp changes along the edge of the dynamic object. When custom stencil value is different between the current frame ($s$) and the warped frame ($s_w$), these pixels will be marked as invalid, i.e.,

$$\Phi_{\text{stencil}} = \{s_i - s_{w,i} \neq 0\} \tag{8}$$

where $i$ is the pixel index and $\Phi_{\text{stencil}}$ is the set including pixels that are counted as invalid according to stencil value.

Second, there will be self occlusions for some dynamic objects when they are moving or the camera is sliding. To mark out these self occluded regions, we resort to world normal since world normal will probably change in these regions. We calculate the cosine value between current frame's world normal ($\mathbf{n}_w$) and warped frame's world normal ($\mathbf{n}_{ww}$), i.e., $\mathbf{n}_w \cdot \mathbf{n}_{ww}$. If this value is large than a predetermined threshold $T_n$, the corresponding pixel indexed by $i$ in the warped frame is counted as invalid. Specifically,

$$\Phi_{\text{wn}} = \{\mathbf{n}_{w,i} \cdot \mathbf{n}_{ww,i} > T_n\} \tag{9}$$

where $\Phi_{\text{wn}}$ contains invalid pixels marked by differences in world normal.

Our third condition handles invalid pixels due to camera movement. These pixels can be selected out by world position, considering that static objects' world positions keep unchanged in a 3D scene. For a given pixel, let $\mathbf{p}_w$ and $\mathbf{p}_{ww}$ be the world position of current frame and warped frame, respectively. We calculate their distance by $|\mathbf{p}_w - \mathbf{p}_{ww}|$. If this distance is larger than a threshold $T_d$ (which is computed by NoV and depth values), we mark this pixel as invalid.

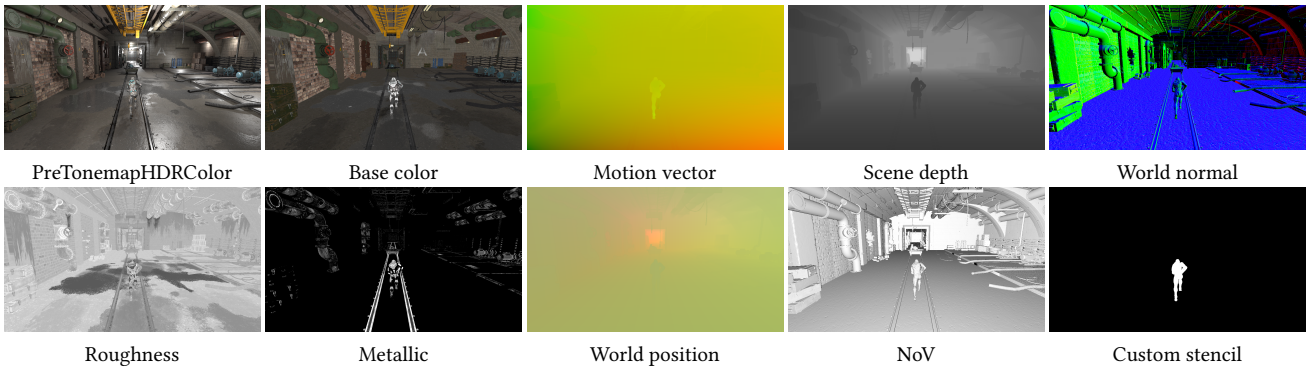



Fig. 11. One example of dumped buffers from Unreal Engine 4. The details and usage of these buffers are explained in Sec. 5.1.

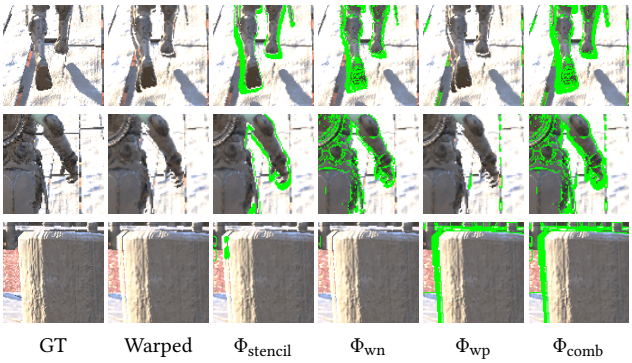



Fig. 12. Demonstration of different hole regions (highlighted in green) marked according to our three conditions. From left to right, we show the ground-truth frame, the warped frame, and hole regions marked out according to $\Phi_{stencil}$, $\Phi_{wn}$, $\Phi_{wp}$, and $\Phi_{comb}$, respectively.

Hence, the invalid pixels in this set $\Phi_{wp}$ are

$$\Phi_{wp} = \{|\mathbf{p}_{w,i} - \mathbf{p}_{ww,i}| > T_d\}. \tag{10}$$

Finally, all invalid pixels in the warped frame are the combination of previous three sets:

$$\Phi_{comb} = \Phi_{stencil} \cup \Phi_{wn} \cup \Phi_{wp}. \tag{11}$$

In Fig. 12, we provide an example to show the invalid pixels marked out by different conditions.

## 6 RESULTS AND COMPARISONS

To demonstrate the effectiveness of our ExtraNet, we compare it against several previous methods. We classify these methods into three groups: methods relying on frame interpolation, methods supporting frame extrapolation and Oculus' ASW technology. For deep learning-based methods, we fine-tune their pre-trained models on our training dataset for the same number of epochs, and use the best performing models obtained during training. Ablation studies are also conducted to assess the key components in our pipeline. All tests are run on a PC equipped with an AMD Ryzen9 3900X CPU and an NVIDIA RTX 3090 GPU. For better comparison, TAA is not enabled.

Table 3. Training setups and time (hours) for different models. MD = `Medieval Docks`. RF = `Redwood Forest`. BK = `Bunker`. WT=`Western Town`. Every model is trained on an NVIDIA RTX 3090 GPU for 150 epochs.

| Models | model1 | model2 | model3 | model4 |
|---|---|---|---|---|
| Training Scene(s) | MD | RF | BK | MD+RF+BK |
| Testing Scene | MD | RF | BK | WT |
| Training Time | 41 | 41 | 35 | 100 |

As aforementioned, we have four datasets in total. Among them three are used for both training and testing, and the rest one is only used for testing. We trained four models as listed in Table 3. The first three models are trained on three scenes individually, while the last model is trained on a combined dataset containing all training examples. The last model is used to test the generalization ability of our network.

### 6.1 Comparisons against Frame Interpolation Methods

Many previous methods leverage frame interpolation to achieve temporal upsampling. In this section, we make comparisons with three typical solutions in this field. Specifically, we respectively compare our ExtraNet to

- *3D Warp* [Mark et al. 1997]: a baseline of forward scattering-based real-time frame interpolation,
- *BSR* (Bidirectional Scene Reprojection [Yang et al. 2011]): a baseline of backward gathering-based real-time frame interpolation, and
- *DAIN* (Depth-Aware video frame INterpolation [Bao et al. 2019]): a state-of-the-art solution of deep learning-based video interpolation.

*Qualitative comparisons.* In Fig. 13, we show the interpolated frames of the above three previous methods, as well as our extrapolated frames. Please note that all interpolation methods require future frames which should be rendered beforehand, while our method only


Fig. 13. Visual comparisons against three frame interpolation methods: 3DWarp [Mark et al. 1997], BSR [Yang et al. 2011], and DAIN [Bao et al. 2019]. Besides high latency, both 3DWarp and BSR may generate inconsistent moving objects and shadows, and will occasionally lose tiny structures. 3DWarp may also cause "rubber sheet" artifacts. DAIN generally achieves higher image quality, but is plagued with blurriness caused by large movements and absence of accurate motion vectors. Our deep learning-based frame extrapolation method avoids these problems and produces physically-plausible results that closely match the ground truth. In the closeups, red arrows highlight some artifacts that are not easily identified and red lines indicate the movements of shadows. See comparisons of animated sequences in the accompanying video.

needs historical frames that have already been rendered. Even so, our deep learning-based method still outperforms these interpolation methods both qualitatively and quantitatively. 3Dwarp [Mark et al. 1997] excels at handling scenes with linear movements, but will fail for dynamic objects with complex movements such as the arm of the running man in the second row of Fig. 13, even if per-pixel motion vectors are used. It may also cause "rubber sheet" artifacts (highlighted in the tenth row) and missing of thin primitives (grasses in the fourth row and twigs in the sixth row) when binary decision in this method is unable to treat complex occlusions. Moreover, 3Dwarp cannot correctly handle view-dependent glossy reflections and dynamic shadows. BSR [Yang et al. 2011] relies on fixed point iteration (FPI) to find the correspondences between two reference frames and uses image blending to fill holes. However, inaccuracy FPI will wash out tiny structures as shown in the eight row of Fig. 13, while image blending will easily overblur high-frequency details and generate ghosting shadows. For DAIN [Bao et al. 2019], a main problem is blurriness caused by large movements and absence of accurate motion vectors, since this method is designed for natural videos. In comparison, our method not only preserves sharp features

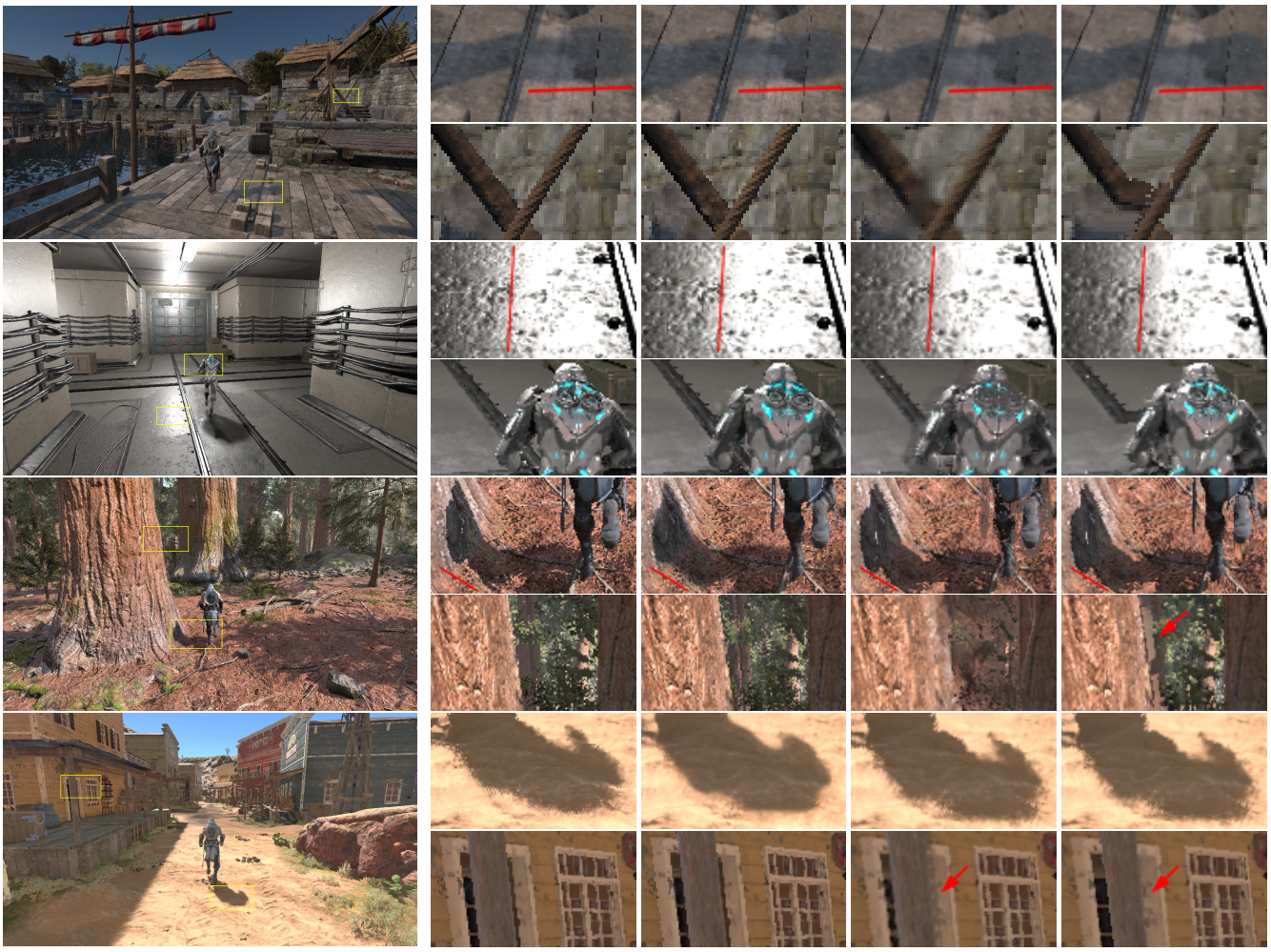



Fig. 14. Visual comparisons against two frame extrapolation methods: GC [Yu et al. 2019] and HIW [Schollmeyer et al. 2017]. GC tends to fill disoccluded regions with average pixel values surrounding the holes, leading to strange silhouettes. HIW may generate distorted structures and confusing colors after image warping. Furthermore, these two methods fail to correct moving shadows and highlights, while our method does particularly well in this aspect. In the closeups, red arrows highlight some artifacts that are not easily identified and red lines indicate the movements of shadows or glossy reflections. See comparisons of animated sequences in the accompanying video.

and tiny geometries, but also generate physically-plausible shadows that closely match the ground-truth results.

*Quantitative comparisons.* To quantitatively analyze different methods, we adopt peak signal-to-noise ratio (PSNR), structural similarity index (SSIM) [Wang et al. 2004] and video multi-method assessment fusion (VMAF) [Netflix 2016] as the error metrics. We evaluate four scenes independently and the results are reported in Table 4. In general, our method outperforms other competitors in terms of every error metric. While DAIN achieves slightly higher PSNR and VMAF values than ours on `Western Town`, its performance degrades dramatically on other scenes, when complex geometries and materials are involved.

## 6.2 Comparisons against Frame Extrapolation Methods

Compared with frame interpolation, frame extrapolation is less studied in the academia. Here, we compare our ExtraNet to two recent methods that can be adapted to support frame extrapolation. *GC* (Gated Convolution [Yu et al. 2019]) is a popular deep learning-based image in-paining method. To enable frame extrapolation, we use our image warping strategy to generate warped images with holes and fill the holes with a GC model fine-tuned on our training dataset. *HIW* (Hybrid Image Warping [Schollmeyer et al. 2017])[2] is tailored for efficient stereo view synthesis. With some trivial

[2]HIW shares the same image synthesize pipeline as that in [Didyk et al. 2010a] and can be viewed as an improvement over it (with an adaptive grid and a better hole filling strategy).

Table 4. Errors in terms of PSNR (dB), SSIM and VMAF on different test sets. Here, we compare our method with 3DWarp [Mark et al. 1997], BSR [Yang et al. 2011], DAIN [Bao et al. 2019], GC [Yu et al. 2019], and HIW [Schollmeyer et al. 2017].

| | | ExtraNet | Interpolation | | | Extrapolation | |
|---|---|---|---|---|---|---|---|
| | | | 3DWarp | BSR | DAIN | GC | HIW |
| PSNR (dB) | MD | **28.18** | 24.31 | 23.97 | 25.73 | 23.68 | 23.23 |
| | RF | **23.50** | 19.22 | 18.70 | 19.95 | 17.71 | 18.45 |
| | BK | **31.19** | 28.13 | 27.00 | 29.47 | 26.30 | 26.95 |
| | WT | 28.97 | 27.83 | 27.22 | **29.41** | 25.22 | 26.84 |
| SSIM | MD | **0.880** | 0.745 | 0.713 | 0.794 | 0.705 | 0.663 |
| | RF | **0.829** | 0.618 | 0.559 | 0.621 | 0.480 | 0.533 |
| | BK | **0.950** | 0.911 | 0.876 | 0.920 | 0.879 | 0.866 |
| | WT | **0.912** | 0.876 | 0.837 | 0.910 | 0.842 | 0.824 |
| VMAF | MD | **84.41** | 76.30 | 72.88 | 81.74 | 70.75 | 71.59 |
| | RF | **83.87** | 75.93 | 73.51 | 76.32 | 63.27 | 72.68 |
| | BK | **91.51** | 87.01 | 82.53 | 88.93 | 81.36 | 83.83 |
| | WT | 89.60 | 86.05 | 81.84 | **90.19** | 79.74 | 82.28 |

modifications, this method is also suitable for frame extrapolation, since stereoscopic rendering shares many similarities with our task.

*Qualitative comparisons.* Fig. 14 provides the qualitative comparisons of different frame extrapolation methods on four scenes. HIW method [Schollmeyer et al. 2017] generate new views using forward warping. Potential hole regions are filled by a multi-scale filtering strategy. Unfortunately, simple filtering may incur artifacts as shown in the even rows of Fig. 14 if the holes are too large. GC [Yu et al. 2019] is originally meant for natural image in-painting. Even the model is fine-tuned on our training dataset, the resulting images are still of low quality. As seen, it tends to fill holes with average pixel values around holes, leading to strange silhouettes. More critically, unlike our method, these two extrapolation methods currently only tackle hole regions, keeping shadings and shadows unchanged. Therefore, they will cause temporal flickering of shadings and shadows in animated sequences.

*Quantitative comparisons.* Quantitative evaluation in terms of PSNR, SSIM and VMAF of different frame extrapolation methods are reported in Table 4. From the comparisons we can see that existing frame extrapolation methods (i.e., GC and HIW) usually produce lower accuracy than interpolation methods, since less information is used. In particular, future frames, which are necessary for frame interpolation, are not involved while extrapolating a new frame. However, our frame extrapolation method still achieves high accuracy even without future frames. It outperforms previous frame extrapolation methods by a large margin.

*User study.* To further compare the quality of different frame extrapolation methods, We conduct a user study on final video sequences. Each time we side-by-side display two video sequences on a LCD screen. One video is generated by our method while the other is randomly chosen from GC, HIW and ground truth (GT). The left / right positions are also randomly placed. The videos are allowed to be played multiple times. 20 participants are asked to determine whether the left-side or right-side videos are more visually pleasant. The pair-wise comparisons of each scene are provided in Fig. 15. As shown, our method is significantly preferred to either GC or HIW among all scenes. For some scenes, participants lean toward the ground truth since our method occasionally generates blurry shadows as we explained in the failure cases. However, for the MD scene participants choose the result of our method even more than the ground truth. This suggests that our method can produce videos that are of equal visual quality with the ground truth.

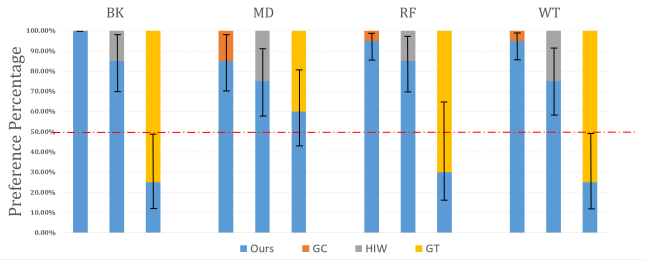



Fig. 15. User study results of our method against GC, HIW and ground truth (GT). 95% confidence intervals are used as error bars.


Fig. 16. Visual comparisons against Oculus' ASW technology [Oculus 2016]. Closeups in red boxes clearly show the artifacts generated by ASW due to incorrect optical flow.

## 6.3 Comparisons against ASW Technology

Oculus' ASW [Oculus 2016] is a mainstream technology in the industry that reduces frame-drops through reprojecting and extrapolating cached frames. Here, we compare our method with a new version of ASW (ASW 2.0) which is integrated in the latest driver of Oculus Rift VR headset. Since little public information is available on the details of the implementation, we can only dump extrapolated frame sequences from the VR headset running ASW 2.0 for visual


Fig. 17. Even on 1080P frames, our method still generates high-quality results that closely match the ground truth, and significantly outperforms previous methods, e.g., DAIN [Bao et al. 2019].

comparison. As shown in the left column of Fig. 16, ASW easily incurs a wide range of incorrect pixels in the image space, since it relies heavily on optical flow to infer motion within the scene. As we known, optical flow estimation often fails at the regions containing complex or repeated geometries. This leads to low-quality image warping and hence noticeable artifacts. The visual results and comparisons in Fig. 16 clearly indicate the superiority of our method.

### 6.4 Test on High-resolution Frames

Since the proposed ExtraNet is fully convolutional, it can naturally handle high-resolution frames. To demonstrate this, we provide some high-resolution examples in Fig. 17. Here, we test our method on 1080P frames. The results clearly show that the extrapolated frames by our method still closely match the ground truth even at a high resolution and are significantly better than those generated by previous methods, e.g., DAIN [Bao et al. 2019].

### 6.5 Analysis of Runtime Performance and Latency

The total runtime of six different methods in comparison is provided in Table 5. The runtime is reported in unit of milliseconds (ms). Since our neural network is specially designed with lightweight gated convolutions and uses NVIDIA TensorRT at 16-bit precision for acceleration, high runtime performance is guaranteed in most scenes. Generally, using lightweight gated convolutions achieves at least 8.50% speedup as compared with using standard gated convolutions. As seen, extrapolating a new frame at 720P resolution with our method is less than 10 ms (including the time for generating G-buffers), which is significantly faster than the conventional per-frame rendering and other deep learning-based methods (i.e., DAIN and GC). This increases the frame rate, especially for very complex scenes. For instance, the MD scene runs at 34 FPS for 720P with RT enabled. Using our method, the frame rate will increase to 54 FPS (1.58×). Although traditional image warping-based methods consume less time in interpolating / extrapolating a new frame, they tend to generate images of much lower quality since rough approximations are usually involved. Moreover, interpolation methods will inevitably introduce additional input latency as future frames should be rendered. This is not desirable in real-time rendering, especially in games where immediate feedback is required.

Table 5. Runtime (milliseconds) comparisons at 720P resolution. Here, we compare our method with 3DWarp [Mark et al. 1997], BSR [Yang et al. 2011], DAIN [Bao et al. 2019], GC [Yu et al. 2019], and HIW [Schollmeyer et al. 2017]. The statistics are measured and averaged over the four test scenes.

| | ExtraNet | Interpolation | | | Extrapolation | |
|---|---|---|---|---|---|---|
| | | 3DWarp | BSR | DAIN | GC | HIW |
| Time | 8.06 | 7.35 | 1.92 | 167 | 91 | 5.83 |

Table 6. Runtime (milliseconds) breakdown of our method at different resolutions. The statistics are measured and averaged over the four test scenes.

| | 480P | 720P | 1080P |
|---|---|---|---|
| G-buffer generation | 1.01 | 1.32 | 1.75 |
| Warping & hole marking | 0.68 | 1.35 | 2.80 |
| Network inference | 2.46 | 5.39 | 11.25 |
| Total | 4.15 | 8.06 | 15.80 |

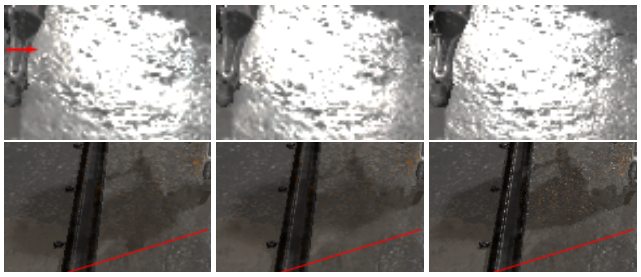
Table 7. Quantitative evaluation in terms of PSNR (dB), SSIM and VMAF on different variants of our methods. HE = History Encoder. OMV = occlusion motion vector.

| | | ExtraNet | w/o $\mathcal{L}_{\text{shade}}$ | w/o HE | w/o OMV |
|---|---|---|---|---|---|
| PSNR (dB) | MD | **28.18** | 27.96 | 27.92 | 27.96 |
| | RF | **23.50** | 22.51 | 22.17 | 22.51 |
| | BK | **31.19** | 29.75 | 30.49 | 30.25 |
| | WT | **28.97** | 28.70 | 28.80 | 28.28 |
| SSIM | MD | **0.880** | 0.877 | 0.877 | 0.878 |
| | RF | **0.829** | 0.798 | 0.791 | 0.799 |
| | BK | **0.950** | 0.937 | 0.944 | 0.941 |
| | WT | **0.912** | 0.910 | 0.908 | 0.900 |
| VMAF | MD | **84.41** | 84.29 | 84.02 | 84.15 |
| | RF | **83.87** | 79.60 | 77.44 | 79.79 |
| | BK | **91.51** | 89.49 | 90.14 | 90.06 |
| | WT | **89.60** | 89.53 | 89.41 | 88.29 |

Table 6 further shows runtime breakdown of our method at three different resolutions. Generally, the runtime performance scales linearly with the number of pixels since our method runs in the image space. It is worth noting that our network can run even faster with delicate engineering, for example, using CUDA and cuDNN optimization rather than TensorRT or writing computer shaders with HLSL. We consider it beyond the scope of our paper and leave it for future work.

### 6.6 Ablation Study

In this section, we conduct ablation studies to evaluate the impact of several key building blocks of our pipeline. The quantitative



w/o History Encoder w/ History Encoder GT

Fig. 18. Comparison between models trained without and with History Encoder. Without History Encoder, glossy reflections (top row, marked by a red arrow) and shadows (bottom row, highlighted by red lines) will not change with respect to movements.

w/o occlusion MV w/ occlusion MV GT

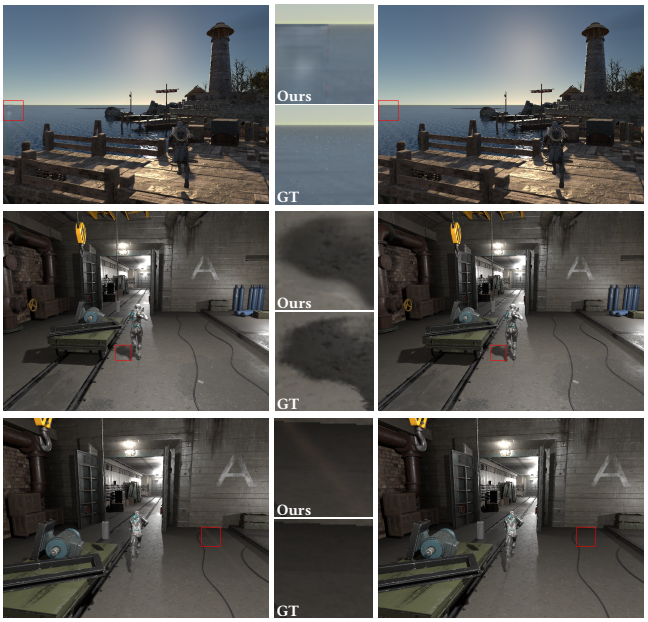
Fig. 19. Comparison between models trained without and with occlusion motion vectors. Ghosting artifacts (along the running man in the left-most image) appear if the model is trained without occlusion motion vectors.

w/o $\mathcal{L}_{\text{shade}}$ w/ $\mathcal{L}_{\text{shade}}$ GT

Fig. 20. Comparison between models trained without and with the shading-augmented loss $\mathcal{L}_{\text{shade}}$. The differences of shadows are highlighted by red lines.

evaluation on different variants of our methods is provided in Table 7.

*Validation of History Encoder.* Unlike most video interpolation methods, we leverage History Encoder instead of widely used optical flow to track movements of shadows and reflections in a scene. This means these movements are implicitly encoded in History Encoder, rather than explicitly represented by optical flow. Without History Encoder, our pipeline fails to capture correct movements of shadows and reflections as shown in Fig. 18, leading to obvious shading latency in these regions.




Ours GT

Fig. 21. Failure cases. Our model currently cannot well handle disocclusion out of screen (top row) and may generate blurry shadows (middle row). It also occasionally incurs ghosting artifacts when the lighting changes suddenly (bottom row).

*Validation of occlusion motion vectors.* Another important design in our ExtraNet is the usage of occlusion motion vectors to warp frames. As aforementioned, occlusion motion vectors help to correct wrong warping by traditional motion vectors in those disoccluded regions. As shown in Fig. 19, improper warping leads to obvious ghosting effects along dynamic objects, such as the running man in the scene.

*Validation of the shading-augmented loss.* In most cases, shading changes only occupy a small portion in a frame. To enlarge their effects, we design the shading-augmented loss $\mathcal{L}_{\text{shade}}$ and use it in our network training. As shown in Fig. 20, this loss is beneficial for capturing plausible shading changes (e.g., moving shadows), avoiding flickering artifacts in animations.

## 7 DISCUSSION AND LIMITATION

*Failure cases.* While our network produces plausible results, there are a few cases where our method does not yet perform very well. Fig. 21 provides a collection of such failure cases.

First, we use both the traditional and disocclusion motion vectors, and train our network to determine which one to learn from in various cases. However, it is possible that neither the traditional motion vectors nor the disocclusion motion vectors work well. In this case, our network would predict wrong information. A similar failure case is usually observed around the border of an image (shown in the top row of Fig. 21), where new contents get in but

was out of the image plane, thus no motion vectors would help in this case.

Second, since we use history frames and train the network to predict shadows, it is essentially performing extrapolations of shadows. This is significantly more difficult than shadow interpolation [Sun et al. 2020]. Our network has successfully learned to predict the movement of shadows, but may sometimes result in overblurred shadow boundaries (shown in the middle row of Fig. 21). Similar artifacts may also happen for highlights and glossy reflections.

Third, our method may generate ghosting artifacts when the lighting in the scene changes suddenly. This is demonstrated in the bottom row of Fig. 21. The ghosting artifact stems from a rapidly flickering light beam in the previous frame which is memorized by our network and improperly retained in the current frame. For such an extreme case, most temporal methods will fail.

*Orthogonal techniques.* There are a few work that are not designed for the extrapolation purpose, but are likely to be directly combined with our method and be potentially helpful.

The first kind of such techniques is the spatial upsampling line of work. This includes TAAU, DLSS and Neural Supersampling [Xiao et al. 2020]. Since our method works with non-antialiased G-buffers, it would be perfect to combine our method with such spatial upsampling work, so that our temporal supersampling will be performed on a low resolution before spatial upsampling. This would result in a much faster performance in both G-buffer generation and network inference. The upsampling approaches do not have to be modified much, except to potentially weigh more on the samples from actual frames rather than extrapolated frames, since the extrapolated shading is after all hallucinated.

The second kind of orthogonal work is about commercial latency reduction techniques, such as NVIDIA Reflex, as well as related approaches such as NVIDIA G-Sync. Such methods are orthogonal because they work either after a frame is rendered (reorganization), or by discarding frames before rendering calls (early termination). With these methods, users will feel even less input latency / lag, especially combined with our extrapolation method.

*Further accelerations on inference.* It is worth noting that the inference time of our network is far from optimized. Currently, available network inference tools, such as PyTorch, ONNX and TensorRT, are mostly designed for efficient evaluations of networks *in large batches of input*, mostly for machine learning and computer vision purposes. However, we only need fast inference once at a time. For this purpose, our current TensorRT implementation can only use the GPU at around 25%. Therefore, a specialized and efficient CUDA implementation that fully utilizes tensor cores (potentially built on cuDNN and cuBLAS) would be able to further boost our performance. Since generating G-buffers for extrapolated frames requires additional computational cost, it is also an interesting future topic to avoid this process to further accelerate the whole pipeline, while still preserving high image quality.

## 8 CONCLUSIONS AND FUTURE WORK

We have presented ExtraNet, an efficient neural network that predicts accurate shading results on an extrapolated frame, to minimize both the performance overhead and the latency. With the help of the rendered auxiliary geometry buffers of the extrapolated frame, and the temporally reliable motion vectors, we train our ExtraNet to perform two tasks simultaneously: irradiance in-painting for regions that cannot find historical correspondences, and accurate ghosting-free shading prediction for regions where temporal information is available. We present a robust hole-marking strategy to automate the classification of these tasks, as well as the data generation from a series of high-quality production-ready scenes. Finally, we use light-weight gated convolutions to enable fast inference. As a result, our ExtraNet is able to produce plausibly extrapolated frames without easily noticeable artifacts, delivering a 1.5× to near 2× increase in frame rates with minimized latency in practice.

In the future, it would be immediately beneficial to combine our ExtraNet with the aforementioned orthogonal techniques. It would also be interesting to train multiple networks to extrapolate more than 1 frames consecutively, and see if the quality is still acceptable while having an even higher frame rate. It is also worth exploring the possibility of an inverse application: using our proposed method (essentially 0 SPP rendering) to help with real-time ray tracing and denoising (using at least 1 SPP). This may lead to further acceleration and potential adaptive sampling insights.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable suggestions. We also would like to thank Zijing Zong and Fan Huang for their assistance with the user study and video. This work was supported by NSFC (62032011 and 61972194), as well as gift funds from Adobe, Dimension 5 and XVerse to L. Yan.

## REFERENCES

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* (Los Angeles, California) *(SIGGRAPH '10)*. Association for Computing Machinery, Article 16, 1 pages.

Simon Baker, Stefan Roth, Daniel Scharstein, Michael J. Black, J.P. Lewis, and Richard Szeliski. 2007. A Database and Evaluation Methodology for Optical Flow. In *2007 IEEE 11th International Conference on Computer Vision*. 1–8.

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Trans. Graph.* 36, 4 (July 2017), 97:1–97:14.

Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. 2019. Depth-Aware Video Frame Interpolation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3698–3707.

Wenbo Bao, Wei-Sheng Lai, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. 2021. MEMC-Net: Motion Estimation and Motion Compensation Driven Neural Network for Video Interpolation and Enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 3 (2021), 933–948.

Huw Bowles, Kenny Mitchell, Robert Sumner, Jeremy Moore, and Markus Gross. 2012. Iterative Image Warping. *Computer Graphics Forum* 31 (05 2012), 1.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4 (July 2017), 98:1–98:12.

Gyorgy Denes, Kuba Maruszczyk, George Ash, and Rafał K. Mantiuk. 2019. Temporal Resolution Multiplexing: Exploiting the limitations of spatio-temporal vision for more efficient VR rendering. *IEEE Transactions on Visualization and Computer Graphics* 25, 5 (2019), 2072–2082.

Piotr Didyk, Elmar Eisemann, Tobias Ritschel, Karol Myszkowski, and Hans-Peter Seidel. 2010a. Perceptually-motivated Real-time Temporal Upsampling of 3D Content for High-refresh-rate Displays. *Computer Graphics Forum* 29, 2 (2010), 713–722.

Piotr Didyk, Tobias Ritschel, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel. 2010b. Adaptive Image-space Stereo View Synthesis. In *Vision, Modeling, and Visualization (2010)*. The Eurographics Association.

Epic Games. 2018. Unreal Engine 4.19: Screen Percentage with Temporal Upsample. https://docs.unrealengine.com/en-US/Engine/Rendering/ScreenPercentage/index.html. Accessed in August 2019.

Denis Fortun, Patrick Bouthemy, and Charles Kervrann. 2015. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding* 134 (2015), 1 – 21. Image Understanding for Real-world Distributed Video Networks.

Jie Guo, Mengtian Li, Quewei Li, Yuting Qiang, Bingyang Hu, Yanwen Guo, and Ling-Qi Yan. 2019. GradNet: Unsupervised Deep Screened Poisson Reconstruction for Gradient-Domain Rendering. *ACM Trans. Graph.* 38, 6, Article 223 (Nov. 2019), 13 pages.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2019. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 558–567.

Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. 2020. RIFE: Real-Time Intermediate Flow Estimation for Video Frame Interpolation. *arXiv preprint arXiv:2011.06294* (2020).

Huaizu Jiang, Deqing Sun, Varan Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. 2018. Super SloMo: High Quality Estimation of Multiple Intermediate Frames for Video Interpolation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9000–9008.

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

Thomas Leimkühler, Hans-Peter Seidel, and Tobias Ritschel. 2017. Minimal Warping: Planning Incremental Novel-view Synthesis. *Computer Graphics Form (Proc. EGSR)* 36, 4 (2017).

Edward Liu. 2020. DLSS 2.0 - Image Reconstruction for Real-Time Rendering with Deep learning. In *Game Developers Conference*.

Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. 2018. Image Inpainting for Irregular Holes Using Partial Convolutions. In *The European Conference on Computer Vision (ECCV)*.

Hongyu Liu, Bin Jiang, Yibing Song, Wei Huang, and Chao Yang. 2020. Rethinking Image Inpainting via a Mutual Encoder-Decoder with Feature Equalizations. In *Computer Vision – ECCV 2020*. Springer International Publishing, Cham, 725–741.

Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. 2017. An efficient denoising algorithm for global illumination. *High Performance Graphics* 10 (2017), 3105762–3105774.

William R. Mark, Leonard McMillan, and Gary Bishop. 1997. Post-Rendering 3D Warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (Providence, Rhode Island, USA) *(I3D '97)*. Association for Computing Machinery, New York, NY, USA, 7–ff.

Joerg H. Mueller, Thomas Neff, Philip Voglreiter, Markus Steinberger, and Dieter Schmalstieg. 2021. Temporally Adaptive Shading Reuse for Real-Time Rendering and Virtual Reality. *ACM Trans. Graph.* 40, 2, Article 11 (April 2021), 14 pages. https://doi.org/10.1145/3446790

Netflix. 2016. Toward a practical perceptual video quality metric. https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652.

Simon Niklaus and Feng Liu. 2020. Softmax Splatting for Video Frame Interpolation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Oculus. 2016. Asynchronous SpaceWarp (ASW). https://developer.oculus.com/blog/asynchronous-spacewarp//.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

AJ Piergiovanni and Michael S. Ryoo. 2019. Representation Flow for Action Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and Hans-Peter Seidel. 2016. Proxy-guided Image-based Rendering for Mobile Devices. *Computer Graphics Forum* 35, 7 (2016), 353–362.

Yurui Ren, Xiaoming Yu, Ruonan Zhang, Thomas H. Li, Shan Liu, and Ge Li. 2019. StructureFlow: Image Inpainting via Structure-Aware Appearance Flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Marcel Santana Santos, Tsang Ing Ren, and Nima Khademi Kalantari. 2020. Single Image HDR Reconstruction Using a CNN with Masked Features and Perceptual Loss. *ACM Trans. Graph.* 39, 4, Article 80 (July 2020), 10 pages.

Daniel Scherzer, Lei Yang, Oliver Mattausch, Diego Nehab, Pedro V. Sander, Michael Wimmer, and Elmar Eisemann. 2012. Temporal Coherence Methods in Real-Time Rendering. *Comput. Graph. Forum* 31, 8 (Dec. 2012), 2378–2408.

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*. 1–12.

Andre Schollmeyer, Simon Schneegans, Stephan Beck, Anthony Steed, and Bernd Froehlich. 2017. Efficient Hybrid Image Warping for High Frame-Rate Stereoscopic Rendering. *IEEE Transactions on Visualization and Computer Graphics* 23, 4 (2017), 1332–1341.

Pradeep Sen, Matthias Zwicker, Fabrice Rousselle, Sung-Eui Yoon, and Nima Khademi Kalantari. 2015. Denoising Your Monte Carlo Renders: Recent Advances in Image-space Adaptive Sampling and Reconstruction. In *ACM SIGGRAPH 2015 Courses* (Los Angeles, California) *(SIGGRAPH '15)*. 11:1–11:255.

Eli Shechtman, Alex Rav-Acha, Michal Irani, and Steve Seitz. 2010. Regenerative Morphing. In *IEEE Conference on Computer VIsion and Pattern Recognition (CVPR)*. San-Francisco, CA.

Karen Simonyan and Andrew Zisserman. 2014. Two-Stream Convolutional Networks for Action Recognition in Videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'14)*. MIT Press, Cambridge, MA, USA, 568–576.

Josef Spjut, Ben Boudaoud, Kamran Binaee, Jonghyun Kim, Alexander Majercik, Morgan McGuire, David Luebke, and Joohwan Kim. 2019. Latency of 30 Ms Benefits First Person Targeting Tasks More Than Refresh Rate Above 60 Hz. In *SIGGRAPH Asia 2019 Technical Briefs* (Brisbane, QLD, Australia) *(SA '19)*. Association for Computing Machinery, New York, NY, USA, 110–113.

Tiancheng Sun, Zexiang Xu, Xiuming Zhang, Sean Fanello, Christoph Rhemann, Paul Debevec, Yun-Ta Tsai, Jonathan T Barron, and Ravi Ramamoorthi. 2020. Light stage super-resolution: continuous high-frequency relighting. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–12.

Thijs Vogels, Fabrice Rousselle, Brian McWilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with kernel prediction and asymmetric loss functions. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–15.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

Xiaoyu Xiang, Yapeng Tian, Yulun Zhang, Yun Fu, Jan P. Allebach, and Chenliang Xu. 2020. Zooming Slow-Mo: Fast and Accurate One-Stage Space-Time Video Super-Resolution. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3370–3379.

Kai Xiao, Gabor Liktor, and Karthik Vaidyanathan. 2018. Coarse Pixel Shading with Temporal Supersampling. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (Montreal, Quebec, Canada) *(I3D '18)*. Association for Computing Machinery, New York, NY, USA, Article 1, 7 pages.

Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural Supersampling for Real-Time Rendering. *ACM Trans. Graph.* 39, 4, Article 142 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392376

Wei Xiong, Jiahui Yu, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. 2019. Foreground-Aware Image Inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Lei Yang, Shiqiu Liu, and Marco Salvi. 2020. A Survey of Temporal Antialiasing Techniques. *Computer Graphics Forum* 39, 2 (2020), 607–621.

Lei Yang, Diego Nehab, Pedro V. Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. 2009. Amortized Supersampling. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–12.

Lei Yang, Yu-Chiu Tse, Pedro V Sander, Jason Lawrence, Diego Nehab, Hugues Hoppe, and Clara L Wilkins. 2011. Image-based bidirectional scene reprojection. In *Proceedings of the 2011 SIGGRAPH Asia Conference*. 1–10.

Zili Yi, Qiang Tang, Shekoofeh Azizi, Daesik Jang, and Zhan Xu. 2020. Contextual Residual Aggregation for Ultra High-Resolution Image Inpainting. , 7505-7514 pages.

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. 2018. Generative Image Inpainting With Contextual Attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. 2019. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE International Conference on Computer Vision*. 4471–4480.

Jiyang Yu and Ravi Ramamoorthi. 2020. Learning Video Stabilization Using Optical Flow. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8156–8164.

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

Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. 2015. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. *Computer Graphics Forum (Proceedings of EGSR)* 34, 4 (June 2015).