FASSET: Frame Supersampling and Extrapolation using Implicit Neural Representations of Rendering Contents

Haoyu Qin, Haonan Zhang, Jie Guo(✉), Ming Yang, Wenyang Bai, and Yanwen Guo

Nanjing University, Nanjing, China
hao19981208@gmail.com,
{502022330066,502022330060}@smail.nju.edu.cn,
{guojie,wyb,ywguo}@nju.edu.cn

Abstract. Despite recent advances in ray tracing hardwares, ray budgets are still limited for many rendering applications, especially when global illumination is enabled. This typically results in undersampling, which manifests as low resolution and low frame-rate when displaying rendering contents. Previous works address this issue by either supersampling a low-resolution input or extrapolating new frames to increase the frame-rate. We introduce a unified and learning-based framework (dubbed FASSET) to conduct frame supersampling and extrapolation jointly, thus significantly reduce the number of pixels that are actually shaded with heavy burden. To handles two tasks simultaneously, we propose implicit neural representations for rendering contents, from which arbitrary-sized frames can be generated using latent features extracted from input low-resolution frames and some auxiliary buffers (e.g., G-buffers). By feeding them with properly warped frames, frame extrapolation with high output resolutions can be achieved as well. Since the implicit neural representations are naturally continuous and their weights are shared across all frames for a given scene, temporal coherence is largely preserved. The proposed framework allows us to only generate $1/8$ pixels every two frames, thus improving the frame-rate to a maximum of $4\times$.

Keywords: Supersampling · Extrapolation · Neural representations · Real-time rendering.

1 Introduction

Recent decades have witnessed dramatic increase of pixel resolutions, and refresh rates in modern displays and this trend seems unlikely to stop in the near future, driven by the enormous demand from virtual reality systems, AAA games, etc. On the other hand, many real-time rendering engines still only afford the generation of relatively low-resolution and low-frame-rate contents due to a limited computing power budget. Moreover, with the emergence of hardware-accelerated APIs, the gaming industry has huge appetite for real-time ray tracing to ensure both high fidelity and high performance

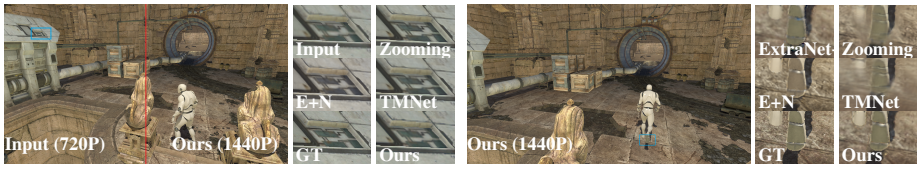

Fig. 1. Visual comparisons of FASSET with other state-of-the-art methods. The left panel shows the comparisons of frame supersampling and the right panel shows the comparisons of frame extrapolation. Closeups highlight the superiority of our method as compared with its competitors, including an extension of ExtraNet [17] (ExtraNet+), ExtraNet+NSRR [61], Zooming-Slow-Mo [59] and TMNet [62].

in rendering contents generation [50, 5, 19]. The heavy pixel workload brought by ray tracing presents a new and great challenge for existing real-time rendering pipelines.

An important and attractive strategy to reduce shading costs is to exploit inter-frame coherence, since most shading results are spatially or temporally coherent [51, 65]. This leaves space for sparse sampling and reuse of neighboring pixel values by gathering samples across space and time, without increasing the total number of samples. Many previous approaches apply such spatially and/or temporally coherent information for anti-aliasing or spatial upsampling, including TAA (Temporal Anti-Aliasing) [66, 60, 65], TAAU (Temporal Anti-Aliasing Upsample) [16], TRM (Temporal Resolution Multiplexing) [12], DLSS (Deep Learning SuperSampling) [28] and NSRR (Neural Supersampling for Real-time Rendering) [61]. These techniques have raised great attention in performance-critical graphical applications in industry. Recently, Guo et al. [17] proposed ExtraNet that leverages temporal information for frame extrapolation, achieving near 2x increase in frame-rates with almost zero latency.

In this paper, we take a step further by applying spatially and temporally coherent information for joint frame supersampling and extrapolation, thus significantly lowering the shading costs. This is an extremely daunting task since only a very small portion of rendering contents are available in this setting. To solve this highly ill-posed problem, we propose *FASSET* (FrAme SuperSampling and ExTrapolation), a unified framework that leverages deep neural networks for upscaling a low-resolution, low-frame-rate video sequence in both space and time. FASSET first extracts deep feature maps from existing low-resolution frames and several cheap G-buffers, through a U-Net [48] style convolutional neural network (CNN). From these deep feature maps, FASSET then learns implicit neural representations to map an image coordinate, conditioned on several latent codes extracted from the deep feature maps around the coordinate, to the corresponding RGB value. Since the coordinates are continuous, the output frames can be presented in arbitrary resolutions in theory. By learning implicit neural representations from properly warped historical frames, the proposed FASSET is also allowed to achieve efficient frame extrapolation, without a noticeable loss in the extrapolated frame's visual quality.

In summary, the main contributions of this paper are:

– a continuous approximation of the rendering contents based on implicit neural representations, named *RenderINR*,

– a lightweight CNN architecture for reliable deep feature maps extraction from existing frames and G-buffers,
– a unified framework that supports joint frame supersampling and extrapolation, enabling a huge reduction of shading costs in rendering while still guaranteeing high image quality.

2 Related work

Frame supersampling/superresolution. Frame supersampling (or superresolution) aims to recover an aliasing-free and (potentially) high-resolution frame from its low-resolution counterpart. Traditional methods resample the historical frames with the help of the accurate motion vectors and some form of temporal reprojection operations [66, 65]. To eliminate the artifacts raised by the shading and visibility changes between adjacent frames, some heuristic neighborhood clamping strategies have been developed to correct the historical samples [25, 49]. To reduce noise, spatial-temporal filters are designed based on frequency analysis of light transport [64] or variance estimation over time [52]. Despite the high efficiency, these heuristic methods are prone to errors and thus easily incur ghosting and other artifacts including loss of details, temporal lag and residual noise in the resultant frames. In the past several years, convolutional neural networks offer a promising alternative for frame supersampling [61, 28, 10, 18]. Xiao et al. [61] proposed a neural supersampling (NSRR) method which combines dense motion vectors and depth with a typical encoder-decoder network. This network takes four consecutive frames as input and predicts up to 4×4 upsampled frames with high fidelity and temporal stability. Nvidia has released deep-learned supersampling (DLSS) [28] that upsamples low-resolution frames with neural networks in real time. Intel also developed a learning-based frame upsampling technique named XeSS [10]. Due to their impressive performance, these learning-based solutions are replacing traditional supersampling and upsampling techniques in industry.

Inter-frame prediction. Inter-frame prediction is a straightforward way to reduce the redundancy within video frames. In general, there are two types of techniques, namely frame interpolation and frame extrapolation. The goal of frame interpolation is to synthesize new frames in the middle of two adjacent frames. Frame interpolation methods can be generally subdivided into four categories: phase-based [34, 33], flow-based [23, 55, 38, 39, 2, 63, 4], kernel-based [27, 40], and direct prediction with feed-forward neural networks [31, 9, 24, 30, 46]. Among them, flow-based methods currently achieve the best performance, especially for rendering contents which provide accurate flow information, i.e., motion vectors [13, 14, 67]. Accurate motion vectors facilitate the reuse of shading information across frames through image warping techniques [32, 13, 14, 47, 53]. Yang et al. [67] interpolated a pair of consecutive rendered frames with bidirectional reprojection. Bowles et al. [3] established a general framework for backward image warping using fixed point iteration. Compared with frame interpolation, frame extrapolation, which predicts future frames from only the past ones, is less studied in either computer graphics or vision. Guo et al. [17] recently trained a deep neural network to perform frame extrapolation, achieving real-time low-latency rendering. There

are some studies in computer vision that achieve spatial super-resolution and temporal interpolation simultaneously for natural videos [62, 8, 59]. However, unifying spatial super-resolution and temporal extrapolation is far less exploited. Nvidia recently announced DLSS 3.0 which combines DLSS super-resolution and frame generation [41]. New frames are generated by optical flow based interpolation, and the Reflex low-latency technology is adopted to minimize the temporal latency caused by interpolation.

Implicit neural representations. Implicit neural representations, which typically parameterize signals using MLP, have been shown to offer competitive advances over explicit representations. In recent years, they have been successfully applied to model complex signals like images [15], videos [6], 3D shapes [44, 29], 3D scenes [22], textures [43] and radiance fields [35]. These neural representations are differentiable and generally more compact compared to canonical representations [54]. They allow discrete signals (such as meshes, voxels and point clouds) to be treated as continuous over bounded domains, since they naturally support smooth interpolations to unseen coordinates. By projecting the coordinates into a higher dimensional space via a positional embedding prior to the MLP, implicit neural representations are capable of conveying fine details [56]. In this paper, we apply implicit neural representations to rendering contents by modeling each rendered frame as a continuous function parameterized by MLP. This allows us recover dense contents from sparse samples, in both spatial and temporal domains.

3 Methodology

While previous work typically views frame supersampling and extrapolation as two separate problems, we solve them jointly with the proposed FASSET framework and thus significantly reduce the overall shading costs. In this section, we describe the details of FASSET.

3.1 Motivation and overview

In most rendering engines, shading usually constitutes a significant part of the workload, especially when global illumination effects are enabled. Therefore, there is always a trade-off between visual quality and runtime performance (frame-rate) for real-time graphical applications. On the other hand, many studies have shown that shading results are spatially or temporally coherent [51, 65]. In viewing of this, the essential motivation behind our FASSET is to significantly reduce the number of pixels that should be actually shaded and leverage light-weight image-space techniques to increase the frame-rate, without sacrificing too much visual quality.

To this end, we opt to render low-resolution frames with advanced shading algorithms. From each frame i with a resolution of $w \times h$, our FASSET conducts two tasks simultaneously:

- performing spatial supersampling to increase frame i's output resolution from $w \times h$ to $2w \times 2h$,

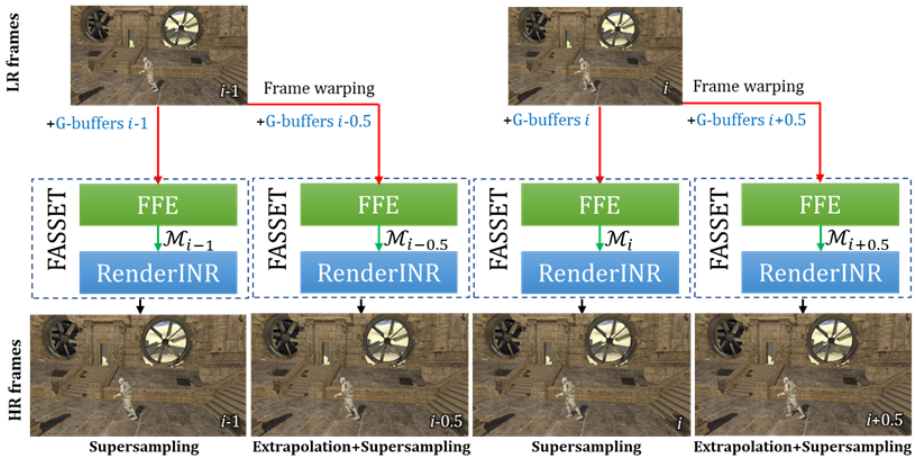

Fig. 2. High-level overview of our method. FFE extracts reliable feature maps from low-resolution frames and G-buffers, while RenderINR handles the query at arbitrary spatial position with the help of the extracted feature maps. Running FFE and RenderINR sequentially allows us to achieve both frame supersampling and extrapolation, thus significantly improving the frame-rate.

– extrapolating a new frame $i + 0.5$ with the output resolution of $2w \times 2h$.

Under this setting, only $1/8$ pixels are actually shaded every two frames, while others are inferred in the image space. This avoids full and dense shading in a frame sequence and enables a huge reduction of shading costs in rendering. The usage of FASSET in a typical rendering engine is demonstrated in Fig. 2.

The first task, supersampling a rendered frame, is well-studied in computer graphics [61, 18]. Aided by cheap G-buffers, such as albedo, normal and depth, we can generate high-resolution frames with high visual quality. In comparison, high-resolution frame extrapolation, which contains both frame extrapolation and supersampling, is a rarely explored field. It seems that we could first perform low-resolution frame extrapolation (e.g., using ExtraNet [17]) and then supersample the frame, or vice versa. However, conducting two tasks sequentially will cause large computational overhead and also easily yield inaccurate results due to error accumulation. To lower the cost and guarantee high image quality, we choose to jointly extrapolate and supersample the frame in our FASSET, using a deep learning-based method.

The proposed FASSET consists of two modules: a CNN-based *frame feature extractor* (FFE) and an implicit neural representation (dubbed *RenderINR*) for rendering contents. The goal of FFE is to generate feature maps to capture the spatial and temporal relationships within the frames. RenderINR, which is realized by coordinate-based multi-layer perceptron (MLP), is designed to decode each input code from the generated feature map (as well as some other auxiliary information) to RGB values. As a continuous rendering frame representation, RenderINR can generate arbitrary-resolution frames in theory.

3.2 Implicit neural representations of rendering contents

Given a rendered frame sequence with limited spatial resolution and frame-rate, we would like to construct a continuous representation for each frame, including the potentially extrapolated frame. This representation interprets an arbitrary spatial coordinate $\mathbf{c}$ into the pixel value p. To achieve this goal, we introduce implicit neural representations of rendering contents (RenderINR), which can be queried at an arbitrary spatial coordinate $\mathbf{c}$, enabling continuous spatial supersampling.

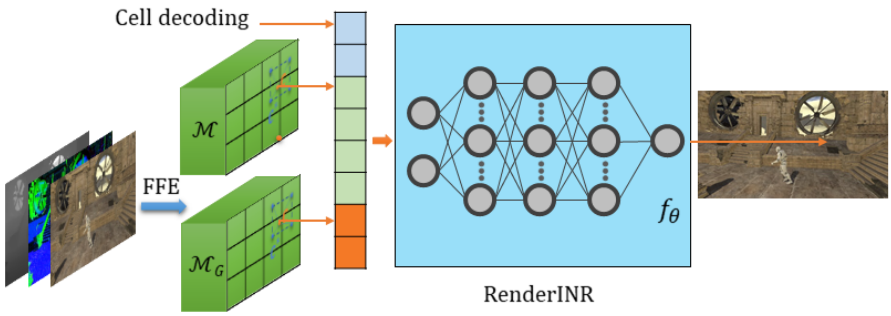

Fig. 3. Illustration of RenderINR. Input of RenderINR includes two feature maps produced by FFE and some coordinate-related information.

RenderINR is realized by fully-connected MLP (denoted as f_θ with θ being its parameters) and shared across all frames. Its input includes latent codes z and z_g, as well as some coordinate-related information. Code z is extracted from the feature map $\mathcal{M}$ and z_g is from $\mathcal{M}_G$. $\mathcal{M}$ and $\mathcal{M}_G$ are both generated by FFE. Specifically, the pixel value at a queried coordinate $\mathbf{c}$ is evaluated as

$$p(\mathbf{c}) = f_\theta(z^*(\mathbf{c}), z_g^*(\mathbf{c}), a) \tag{1}$$

where z^* and z_g^* is the latent code in $\mathcal{M}$ and $\mathcal{M}_G$, extracted by bilinear sampling. $\mathbf{c}$ is its corresponding coordinate in the 2D image space. Inspired by [7], we also incorporate the pixel size a into the input. This is beneficial for presenting the continuous representation in a give high resolution. f_θ has parameters $\theta = \{\mathbf{W}_i, \mathbf{b}_i\}$ containing weights $\mathbf{W}_i$ and biases $\mathbf{b}_i$ of each layer i.

In our current implementation, we use an MLP with three hidden layers that have a width of 128 neurons, ReLU activation functions on the hidden layers, and a linear output layer. This compact and light-weight design of MLP leads to a small number of trainable parameters (less than 40K), offering fast training/inference and a good generalization behavior.

3.3 Frame feature extractor

MLPs alone have no capability to model the spatial relationships within the represented signals, thus are not performant for image synthesis applications such as image supersampling. The situation becomes worse for the task of frame extrapolation, due to the

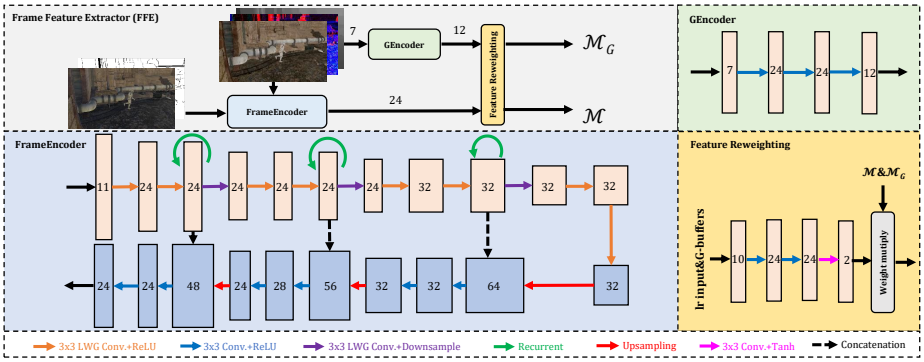

Fig. 4. The detailed network architecture of FFE. Input frames and G-buffers are fed into FrameEncoder to extract a temporal feature map $\mathcal{M}$ with a resolution of $w \times h$. G-buffers are fed to GEncoder to extract another feature map $\mathcal{M}_G$. Feature reweighting module takes original frames and G-buffers as input, and outputs corresponding weights for $\mathcal{M}$ and $\mathcal{M}_G$.

existence of invalid pixels at dis-occluded areas. We address these issues by extracting the feature map $\mathcal{M}$ from the input frames and another feature map from G-buffers. To handle frame supersampling and extrapolation simultaneously, the feature map $\mathcal{M}$ should have:

- the awareness of spatial relationships inside the input frame,
- the guarantee of temporal coherence between adjacent frames, and
- the ability to inpaint disoccluded areas when performing image warping for frame extrapolation.

We leverage a typical CNN architecture to address the above issues since stacked convolutional layers are apt at capturing multi-scale spatial structures. Temporal coherence is enabled by shared weights of the networks across all input frames. By incorporating some light-weight gated (LWG) convolutional layers [68], the CNN-based model is able to inpaint dis-occluded areas after image warping.

Overall pipeline We name our CNN-based model as Frame Feature Extractor (FFE) which contains a FrameEncoder, a GEncoder and a feature reweighting module, as illustrated in Fig. 4. The ultimate goal of FFE is to extract proper feature map $\mathcal{M}$ and $\mathcal{M}_G$ from input frames (warped for the extrapolation task) and some auxiliary buffers (G-buffers and masks indicating holes after frame warping). To this end, we construct a feature map $\mathcal{M}$ with FrameEncoder and a G-buffers feature map $\mathcal{M}_G$ with GEncoder. A continuous implicit neural representation, i.e., RenderINR, is then constructed on top of the output feature maps $\mathcal{M}$ and $\mathcal{M}_G$.

Frame warping and FrameEncoder Frame warping is required for the task of frame extrapolation. Inspired by ExtraNet [17], we warp frames using traditional motion vectors. In order to remove the influence of ghosting, we multiply the input by a corresponding mask indicating holes after warping. This warped frame, together with the mask and

some G-buffers (depth and normal), are fed into FrameEncoder. For frames that require only supersampling, we do not need to warp the input, but we also multiply the input by the mask. There are two benefits to this method of data processing:

- guaranteeing that the supersampled frames and extrapolated frames have inputs of the same type, enhancing temporal coherence between adjacent frames.
- increasing the number of training examples for image inpainting, augmenting the ability of hole filling.

The masks for supersampling are generated by making holes of warped previous frames.

The detailed network architecture of FrameEncoder is shown in Fig. 4 in blue. This is a typical U-Net architecture that contains skip connections between mirrored layers in the encoder and decoder stacks. This is beneficial for preserving high-frequency details after convolution operations. LWG convolutional layers are employed to fill holes after frame warping. We add recurrent module after three convolution operations in the encoder to reuse the information of history frames. Moreover, we use a residual learning strategy across our pipeline.

GEncoder G-buffers are fed to GEncoder in FFE before RenderINR. The GEncoder consists of three convolutional layers and output a G-buffers feature map $\mathcal{M}_G$. The structure of GEncoder is shown in Fig. 4. There are two benefits to add this module. First, GEncoder will extract more information in G-buffers and provide more details in results. Second, feeding the G-buffers to RenderINR directly will give incorrect information on pixels, leading to artifacts.

Feature reweighting Before feeding $\mathcal{M}$ and $\mathcal{M}_G$ to RenderINR, FFE adds a feature reweighting module to reweight these two feature maps. The structure of this module is shown in in Fig. 4. This module is a 3-layer convolutional neural network, which takes the RGB input and G-buffers as input, generating pixel-wise weighting maps for $\mathcal{M}$ and $\mathcal{M}_G$. Tanh activation function follows last convolution, and then maps the values from (-1, 1) to (0, 10), where 10 is a hyper-parameter.

3.4 Network training

Loss function We train our network using three types of loss functions. The first loss is pixel-wise error between the predicted frame F and ground-truth G. This is the L1 distance between F and G:

$$\mathcal{L}_{l_1} = \frac{1}{n} \sum_i |F_i - G_i| \tag{2}$$

where n is the number of pixels in the frames.

For inpainting in the hole regions, we use the mask loss to enhance the loss of masked pixels by hole masks. This loss is expressed as:

$$\mathcal{L}_{holeMask} = \frac{1}{n - \sum_i h_i} \sum_i |F_i - G_i| \cdot (1 - h_i) \tag{3}$$

where h is the binary mask indicating holes.

The third loss is designed for correcting shading. The loss of hole mask can not include all wrong pixels such as incorrect shadow and highlight. Therefore, we use hard loss to overcome this problem. We select k pixels with top k largest errors from the L1 loss and regard these pixels as regions of incorrect shading results where k is 1/10 of the total pixels. This loss is expressed as:

$$\mathcal{L}_{hardPixels} = \frac{1}{n} \sum_{i \in P_{top-k}} |F_i - G_i| \tag{4}$$

where P_{top-k} is the k largest L1 loss in all pixels. Our final loss function is the summation of all above losses:

$$\mathcal{L} = \mathcal{W}_{l1}\mathcal{L}_{l_1} + \mathcal{W}_{hole}\mathcal{L}_{holeMask} + \mathcal{W}_{hard}\mathcal{L}_{hardPixels} \tag{5}$$

The losses above are all borrowed from ExtraNet [17]. L1 loss ensures the overall quality of results. While hole loss is beneficial for inpainting the holes caused by warping. Different from ExtraNet [17], we set $\mathcal{W}_{hole}$ to 5, because we find this weight can get better results in hole regions. Hard loss is necessary for our method, since shadow and highlight will stay in the original position in previous frames without this loss.

Training details Our FASSET is implemented using the PyTorch framework [45]. Adam optimizer [26] is used for optimization. Specifically, we set mini-batch size as 8, the parameters (β_1, β_2) of Adam optimizer are set as $(0.9, 0.999)$. Every model is trained for 150 epochs. The learning rate is 0.001 initially and decays half every 30 epochs. Before feeding images into our network, logarithm transformation $y = \log(1 + x)$ is applied to HDR images to avoid large values.

4 Experiments

4.1 Dataset

Our training and test datasets are generated using a modified version of Unreal Engine 4. We record several sequences for each scene, and the splitting of training and testing sets as well as some statistics are shown in Table 1. The training and testing sets have no overlap.

Sequential frames are rendered and dumped into the .exr file format. We implement a Compute Shader within the render passes of Unreal Engine 4. For each frame, Unreal Engine 4 generates the following buffers for our dataset:

1. The shading frame before tone mapping (PreTonemapHDRColor).
2. Three kinds of G-buffers used in our network, including albedo, scene depth and world normal.
3. Other auxiliary buffers for data preprocess, including motion vector, world position, NoV (the dot product of world normal and view vector) and customized stencil (a stencil buffer for masking dynamic objects).

Table 1. Statistics of the training and testing datasets used in validating our method. 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
Hideout (HO)	4	4	2838	1400
Factory (FE)	4	3	2423	1248
Temple (TP)	4	4	2840	1162

Note that scene depth, world normal and albedo are rendered in low-resolution for our pipeline. We use high-resolution masks for the calculation of our loss function and feed low-resolution masks into networks, while other buffers in our dataset are rendered in low-resolution. Our method can replace the actual shading pass in Unreal Engine 4. Therefore, post-processing passes like anti-aliasing are still required to generate the final sequence.

When a previous frame is warped to the current frame, some pixels in the warped frame will be invalid. These pixels cause "holes" and should be marked out before feeding into our networks. Inspired by ExtraNet [17], we mark holes from three different dimensions including custom stencil difference ($S_{stencil}$), self occlusion (S_{wn}) and occlusion caused by camera's movement (S_{wp}). The methods of marking holes are the same with ExtraNet [17]. The final mask S combines the these three sets of invalid pixels:

$$S = S_{wp} \cup S_{wn} \cup S_{stencil}. \tag{6}$$

4.2 Baselines and Settings

In the following experiments, we take $2\times$ as the scale factor of frame supersampling. For training sets, we render the frames and corresponding G-buffers at 640×360 (360P) as low-resolution input and 1280×720 (720P) as high-resolution images, respectively. For test sets, the data at 720P is generated as low-resolution frames and the frames of ground truth are rendered at 2560×1440 (1440P).

To demonstrate the effectiveness of our FASSET, we compare it against the state-of-the-arts. DLSS 3 [41] is similar to our method, but it has little available public information. Therefore, we choose NSRR [61] for rendering frame supersampling and ExtraNet [17] for rendering frame extrapolation. They are combined as two two-stage baselines: ExtraNet+NSRR and NSRR+ExtraNet. ExtraNet and NSRR are trained separately in these two baselines. For a fair comparison, we use low-resolution G-buffers in ExtraNet and upsample them to high-resolution for NSRR+ExtraNet. Additionally, we choose two space-time video supersampling methods for comparison: Zooming-Slow-Mo [59] and TMNet [62]. At last, ExtraNet+ is a modified version of ExtraNet [17] which generates extrapolated frames end-to-end at high-resolution space. For ExtraNet+, we zero-padding the input frames and history frames, which are fed to the network of ExtraNet directly. Therefore, ExtraNet+ take high-resolution input and outputs high resolution extrapolated frames which needs to inpaint the holes caused by warping and zero-padding. The models above are all trained on the training datasets with 150

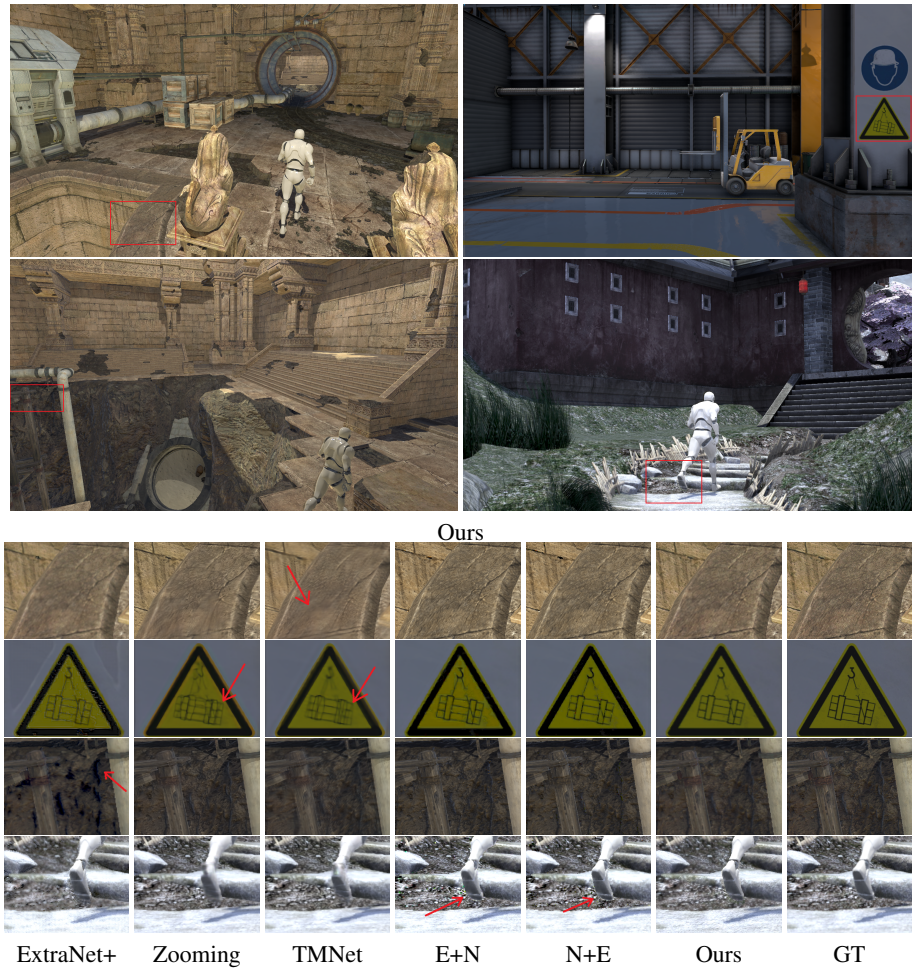

Fig. 5. Comparison on frame extrapolation/interpolation against other render-based and video-based methods. The first two rows are the example frames from our test datasets, and the last four rows are cropped respectively from the red rectangles in the images. From left to right: ExtraNet+, Zooming-Slow-Mo [59], TMNet [62], ExtraNet [17]+NSRR [61], NSRR+ExtraNet, Ours, GT.

epochs for fair comparison. All tests are run on a PC equipped with an NVIDIA RTX 3090 GPU.

Quantitative Comparison To quantitatively analyze these results, we choose signal-to-noise ratio (PSNR), structural similarity index (SSIM) [58] and LPIPS [70] as the error metrics. We evaluate three scenes and the quantitative results are reported in Table 2. As ExtraNet+ generates extrapolated frames only, we do not report its results of frame super-resolution. NSRR is the method for frame supersampling and the same history frames and buffers are fed into network in both Extranet+NSRR and NSRR+ExtraNet.

Therefore, ExtraNet+NSRR and NSRR+ExtraNet have the same results for frame supersampling.

Table 2. Quantitative comparison of PSNR, SSIM and LPIPS on all scenes. Hideout (HO), Temple (TP), Factory Environment (FE) are the names of different scenes. The results on the interpolated/extrapolated frames (the left indicator of /) and the supersampling frames (the right indicator of /) are separately shown in the table.

	Scene	ExtraNet+	NSRR+ExtraNet	ExtraNet+NSRR	Zooming	TMNet	FASSET
PSNR(dB)↑	HO	29.54/*	26.84/33.20	32.22/33.20	30.09/31.66	28.25/32.17	30.62/31.15
	TP	22.94/*	23.98/25.20	24.34/25.20	23.67/26.48	22.58/26.00	24.56/25.46
	FE	28.35/*	26.70/31.04	29.28/31.04	31.60/32.25	31.68/32.46	30.93/31.59
SSIM↑	HO	0.8120/*	0.7159/0.9001	0.8770/0.9001	0.7830/0.9038	0.6631/0.8972	0.8308/0.8503
	TP	0.6889/*	0.8157/0.8809	0.8509/0.8809	0.8228/0.8933	0.7388/0.8870	0.8460/0.8716
	FE	0.8508/*	0.8254/0.9137	0.8930/0.9137	0.9001/0.9215	0.9010/0.9245	0.8968/0.9093
LPIPS↓	HO	0.3068/*	0.2963/0.2080	0.2364/0.2080	0.3257/0.1935	0.4092/0.2043	0.2667/0.2497
	TP	0.4033/*	0.2839/0.2164	0.2467/0.2164	0.2641/0.2062	0.3152/0.2150	0.2499/0.2194
	FE	0.2940/*	0.2760/0.2182	0.2455/0.2182	0.2633/0.2095	0.2696/0.2064	0.2483/0.2172

As seen, we generate better results in both frame supersampling and frame extrapolation, while other methods have a significant performance drop on extrapolated or interpolated frames except ExtraNet+NSRR. However, in terms of metric values between extrapolated and supersampled frames, FASSET shows smaller delta than ExtraNet+NSRR, which presents FASSET can get more stable results than ExtraNet+NSRR.

Among these methods, only ExtraNet+ has similar inference time with FASSET according to Table 3 and its quantitative results are far below FASSET. For other methods, FASSET also have competitive results in some metrics such as extrapolated results on TP. Because FASSET is faster than other methods except ExtraNet+, especially methods for video, there is a trade-off between inference time and quantitative results for FASSET. Despite all this, our results are also very close to best results for other metrics, while our method is at least 5 times faster than these methods.

Qualitative comparison In Fig. 5, we show the qualitative comparisons of different methods for frame extrapolation/interpolation. It is worth noting that Zooming-Slow-Mo and TMNet require future frames which should be rendered beforehand, while historical frames that have already been rendered are needed in FASSET. ExtraNet+ tends to generate artifacts as highlighted in the last three rows in Fig. 5. Because of redundant information from G-buffers, NSRR+ExtraNet and ExtraNet+NSRR generates incorrect color like the yellow triangle sign in the fourth row and ropes tied to wooden stakes in the fifth row. TMNet and Zooming-Slow-Mo are originally designed for slow-motion videos, and their results are plagued with over blurriness caused by large movements.

Fig. 6 show visual comparisons of different methods for frame supersampling. In the last row, we can see that other methods reduce the rutting stains on the yellow guide

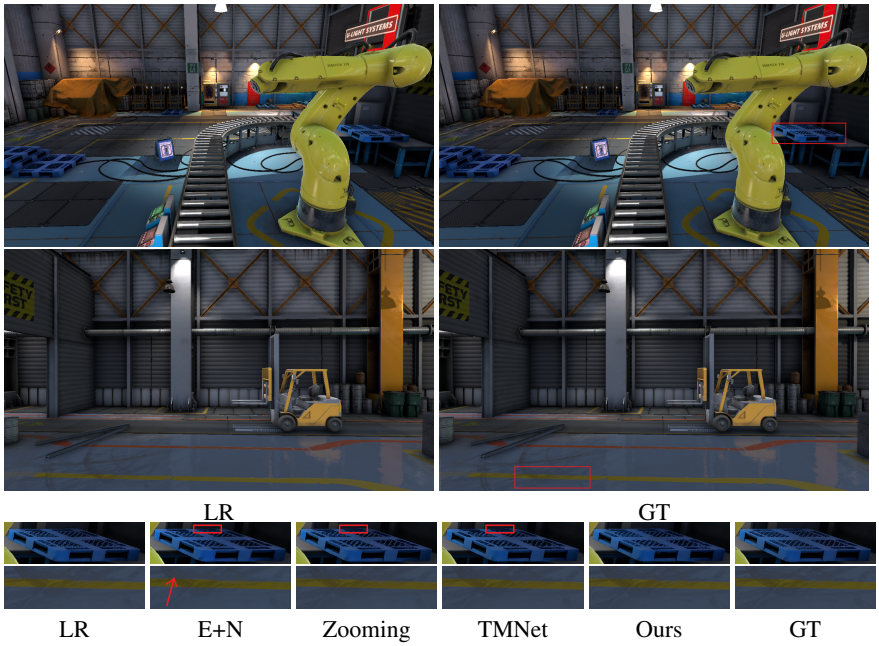

Fig. 6. Comparison on frame supersampling against other render-based and video-based methods. The first two rows are the example frames from test datasets, and the last two rows are cropped respectively from the red rectangles in the images. From left to right: LR, ExtraNet [17]+NSRR [61], Zooming-Slow-Mo [59], TMNet [62], Ours, GT.

belt. In contrast, FASSET can capture useful features from low-resolution images and G-buffers, and have better supersampling performance in details.

4.3 Analysis of runtime performance and model efficiency.

Table 3 lists the computation cost, memory throughput, model parameters and runtime for processing a single frame. NVIDIA TensorRT is used for acceleration when calculating the inference runtime of all models. NSRR+ExtraNet need to extrapolate the frames at high-resolution space and feed data at 1440P into ExtraNet, while ExtraNet+NSRR feed data at 720P into ExtraNet. FASSET needs 20.46ms per frame for network inference and has 0.27M parameters only, which is significantly lower than video methods and basically equal to ExtraNet+ compared to others. Even considering all the benefits from reduced precision computation, the network structures of other methods still take more computation cost than ours. In particular, TMNet and Zooming-Slow-Mo need future rendered frame for interpolation, which causes the extra time latency.

We display runtime breakdown of our method which targets at 1440P resolution in Fig. 7. In practice, we will save more time with two GPUs because the processes of frame supersampling and extrapolation are independent which can run in parallel. This

Table 3. The network computation cost, memory throughput comparison, model parameters and runtime among FASSET and other methods. The metric is measured for the 720P to 1440P. All tests are run on an NVIDIA RTX 3090 GPU.

Method	#Params (M)	Flops (G)	GPU (MiB)	Time (ms)
ExtraNet+	0.10	32	2639	9.94
E+N	0.76	695	15849	131.50
N+E	0.76	836	18608	157.51
TMNet	10.88	25421	17547	1043.94
Zooming	10.29	55860	21296	967.18
FASSET	0.27	216	14433	20.46

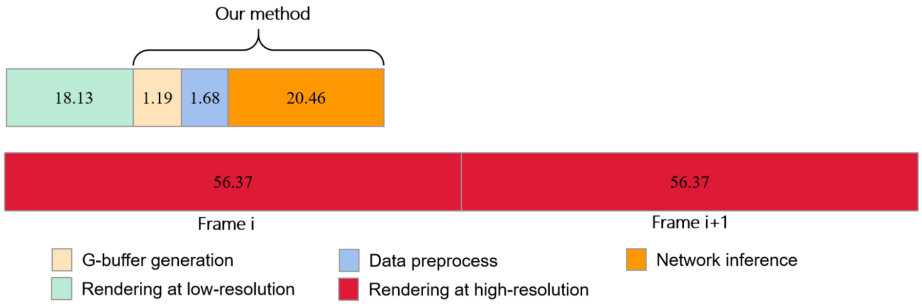

Fig. 7. Runtime (ms) breakdown of our method targeting at 1440P resolution. The statistics are measured and averaged over the all test scenes.

allows us to improve the frame-rate to 3.68x using the reasonable settings of parallel computing. It is also worth noting that FASSET can run faster with delicate engineering, such as using CUDA and cuDNN optimization or compute shaders.

4.4 Ablation study

Validation of GEncoder Before feeding G-buffers to RenderINR, we feed them to GEncoder first for feature extraction. It is beneficial for reproducing the details in images and removing artifacts. Moreover, we do not choose feeding G-buffers directly, which is possible to generate artifacts because G-buffers can not replace pixel colors directly. As shown in Fig. 8, the result with GEncoder has more details compared to the one without GEncoder.

w/o GEncoder w/ GEncoder GT

Fig. 8. Comparison between models trained without/with GEncoder

Validation of mask multiplying on frame supersampling When input frames are supersampled, the encoder has no need to deal with the inpainting problem during training, and it will influence the training of inpainting ability. Therefore, we employ additional masks to supersampled frames to enhance the inpainting ability of encoder. As shown in Fig. 9, without additional masks, our pipeline may fail to inpaint the hole of moving person.

w/o mask w/ mask GT

Fig. 9. Comparison between models trained without/with additional mask

Validation of Extra/Super shared weight of encoder Because networks need to achieve different goals between adjacent frames, there will be instability if we use different weights of encoder for extrapolated frames and supersampled frames. As shown in Fig. 10, shared weights of encoder are beneficial for stability between frames. Without shared weights, adjacent frames have obvious differences in highlights on the pillar.

frame i frame i+1 frame i frame i+1

Fig. 10. Comparison between models trained with/without shared weight of encoder. The left two are generated by our method and the right two are generated by model without shared weight.

Table 4 shows the ablation experiments for above modules. We observe that all of these modules can improve quantitative results, and overall visual quality.

Table 4. Ablation study of GEncoder, additional masks and shared weights on the Hideout (HO) scene. We show the metrics for supersampled frames and extrapolated frames separately.

	PSNR↑	SSIM↑	LPIPS↓
FASSET	30.62/31.15	0.8308/0.8503	0.2667/0.2497
- GEncoder	29.21/30.38	0.8175/0.8365	0.2807/0.2593
- Additional mask	29.88/30.31	0.8212/0.8381	0.2783/0.2536
- Shared weights	29.66/30.44	0.8248/0.8458	0.2772/0.2581

4.5 Limitation

We present a joint framework for both frame supersampling and extrapolation on render contents. As motion information is extracted from motion vectors generated by rendering engines, we can not handle shadows, highlights and glossy reflections well when the corresponding areas have strong motions.

Because of the huge amount of memory and time cost on training, we do not test the generalization ability of FASSET. Employing powerful hardware optimization and network quantization to further decrease the inference time while still preserving high image quality is also an interesting topic. We leave them for future research.

5 Conclusion

In this paper, we have introduced FASSET which integrates an implicit representation of rendering contents and a CNN-based frame feature extractor in a unified end-to-end framework, where the implicit neural representation is adopted in the decoding stage to learn continuous-resolution representation. Frame feature extractor consisting of three parts including frameEncoder, GEncoder and feature reweighting module is used to extract feature maps and solve the inpainting problem, following RenderINR, which is realized by MLP to get high-resolution results. Because our method supports joint frame supersampling and extrapolation and has lightweight network, its inference time is far less than other methods, and also has competitive quantitative results and qualitative results compared with others.

References

1. Anwar, S., Khan, S., Barnes, N.: A deep journey into super-resolution: A survey. ACM Comput. Surv. **53**(3) (may 2020)
2. Bao, W., Lai, W.S., Ma, C., Zhang, X., Gao, Z., Yang, M.H.: Depth-aware video frame interpolation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
3. Bowles, H., Mitchell, K., Sumner, R., Moore, J., Gross, M.: Iterative image warping. Computer Graphics Forum **31**, 1 (05 2012)
4. Briedis, K.M., Djelouah, A., Meyer, M., McGonigal, I., Gross, M., Schroers, C.: Neural frame interpolation for rendered content. ACM Trans. Graph. **40**(6) (dec 2021)
5. Burgess, J.: Rtx on the nvidia turing gpu. IEEE Micro **40**(2), 36–44 (2020)
6. Chen, H., He, B., Wang, H., Ren, Y., Lim, S.N., Shrivastava, A.: NeRV: Neural representations for videos. In: Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems (2021), https://openreview.net/forum?id=BbikqBWZTGB
7. Chen, Y., Liu, S., Wang, X.: Learning continuous image representation with local implicit image function. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8628–8638 (2021)
8. Chen, Z., Chen, Y., Liu, J., Xu, X., Goel, V., Wang, Z., Shi, H., Wang, X.: Videoinr: Learning video implicit neural representation for continuous space-time super-resolution (2022)
9. Choi, M., Kim, H., Han, B., Xu, N., Lee, K.M.: Channel attention is all you need for video frame interpolation. In: AAAI (2020)
10. Chowdhury, H., Kawiak, Robert, R., de Boer, Ferreira, G., Xavier, L.: Intel XeSS – an AI based super sampling solution for real-time rendering. In: Game Developers Conference (2022)
11. Deliot, T., Guinier, F., Vanhoey, K.: Real-time style transfer in unity using deep neural networks. https://blogs.unity3d.com/2020/11/25/real-time-style-transfer-in-unity-using-deep-neural-networks/ (Nov 2020)
12. Denes, G., Maruszczyk, K., Ash, G., Mantiuk, R.K.: Temporal resolution multiplexing: Exploiting the limitations of spatio-temporal vision for more efficient vr rendering. IEEE Transactions on Visualization and Computer Graphics **25**(5), 2072–2082 (2019)
13. Didyk, P., Eisemann, E., Ritschel, T., Myszkowski, K., Seidel, H.P.: Perceptually-motivated real-time temporal upsampling of 3d content for high-refresh-rate displays. Computer Graphics Forum **29**(2), 713–722 (2010)

14. Didyk, P., Ritschel, T., Eisemann, E., Myszkowski, K., Seidel, H.P.: Adaptive Image-space Stereo View Synthesis. In: Vision, Modeling, and Visualization (2010). The Eurographics Association (2010)
15. Dupont, E., Goliński, A., Alizadeh, M., Teh, Y.W., Doucet, A.: COIN: Compression with implicit neural representations (2021), https://arxiv.org/abs/2103.03123
16. Epic Games: Unreal Engine 4.19: Screen percentage with temporal upsample. https://docs.unrealengine.com/en-US/Engine/Rendering/ScreenPercentage/index.html (Mar 2018), accessed in August 2019
17. Guo, J., Fu, X., Lin, L., Ma, H., Guo, Y., Liu, S., Yan, L.Q.: Extranet: Real-time extrapolated rendering for low-latency temporal supersampling. ACM Trans. Graph. **40**(6) (dec 2021)
18. Guo, Y.X., Chen, G., Dong, Y., Tong, X.: Classifier Guided Temporal Supersampling for Real-time Rendering. Computer Graphics Forum (2022). https://doi.org/10.1111/cgf.14672
19. Harada, T.: Hardware-accelerated ray tracing in amd radeon prorender 2.0. https://gpuopen.com/learn/radeon-prorender-2-0/ (2020)
20. Herzog, R., Eisemann, E., Myszkowski, K., Seidel, H.P.: Spatio-temporal upsampling on the gpu. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. pp. 91–98. I3D '10, Association for Computing Machinery, New York, NY, USA (2010)
21. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7132–7141 (2018)
22. Jiang, C.M., Sud, A., Makadia, A., Huang, J., Nießner, M., Funkhouser, T.: Local implicit grid representations for 3d scenes. In: Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) (2020)
23. Jiang, H., Sun, D., Jampani, V., Yang, M.H., Learned-Miller, E., Kautz, J.: Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
24. Kalluri, T., Pathak, D., Chandraker, M., Tran, D.: FLAVR: Flow-agnostic video representations for fast frame interpolation (2021)
25. Karis, B.: High-quality temporal supersampling. SIGGRAPH 2014 Advances in Real-Time Rendering in Games course (2014)
26. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
27. Lee, H., Kim, T., Chung, T.y., Pak, D., Ban, Y., Lee, S.: Adacof: Adaptive collaboration of flows for video frame interpolation. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
28. Liu, E.: Dlss 2.0 - image reconstruction for real-time rendering with deep learning. In: Game Developers Conference (2020)
29. Liu, H.T.D., Williams, F., Jacobson, A., Fidler, S., Litany, O.: Learning smooth neural functions via lipschitz regularization. In: ACM SIGGRAPH 2022 Conference Proceedings. SIGGRAPH '22, Association for Computing Machinery, New York, NY, USA (2022)
30. Liying Lu, Ruizheng Wu, H.L.J.L., Jia, J.: Video frame interpolation with transformer. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2022)
31. Long, G., Kneip, L., Alvarez, J.M., Li, H., Zhang, X., Yu, Q.: Learning image matching by simply watching video. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer Vision – ECCV 2016. pp. 434–450. Springer International Publishing, Cham (2016)
32. Mark, W.R., McMillan, L., Bishop, G.: Post-rendering 3d warping. In: Proceedings of the 1997 Symposium on Interactive 3D Graphics. I3D '97, Association for Computing Machinery, New York, NY, USA (1997)

33. Meyer, S., Djelouah, A., McWilliams, B., Sorkine-Hornung, A., Gross, M., Schroers, C.: Phasenet for video frame interpolation. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 498–507. IEEE Computer Society, Los Alamitos, CA, USA (jun 2018)
34. Meyer, S., Wang, O., Zimmer, H., Grosse, M., Sorkine-Hornung, A.: Phase-based frame interpolation for video. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1410–1418 (2015)
35. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: European conference on computer vision. pp. 405–421. Springer (2020)
36. Mueller, J.H., Neff, T., Voglreiter, P., Steinberger, M., Schmalstieg, D.: Temporally adaptive shading reuse for real-time rendering and virtual reality. ACM Trans. Graph. **40**(2) (apr 2021)
37. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans. Graph. **41**(4) (jul 2022)
38. Niklaus, S., Liu, F.: Context-aware synthesis for video frame interpolation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
39. Niklaus, S., Liu, F.: Softmax splatting for video frame interpolation. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
40. Niklaus, S., Mai, L., Liu, F.: Video frame interpolation via adaptive convolution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
41. Nvidia: Nvidia dlss 3.0. https://www.nvidia.com/en-us/geforce/technologies/dlss/ (2022)
42. Oculus: Asynchronous spacewarp (asw). https://developer.oculus.com/blog/asynchronous-spacewarp// (2016)
43. Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., Geiger, A.: Texture fields: Learning texture representations in function space. In: Proceedings IEEE International Conf. on Computer Vision (ICCV) (2019)
44. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
45. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 8024–8035 (2019), https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
46. Reda, F., Kontkanen, J., Tabellion, E., Sun, D., Pantofaru, C., Curless, B.: FILM: Frame interpolation for large motion. In: European Conference on Computer Vision (ECCV) (2022)
47. Reinert, B., Kopf, J., Ritschel, T., Cuervo, E., Chu, D., Seidel, H.P.: Proxy-guided image-based rendering for mobile devices. Computer Graphics Forum **35**(7), 353–362 (2016)
48. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. pp. 234–241. Springer International Publishing, Cham (2015)
49. Salvi, M.: An excursion in temporal supersampling. Game Developer's Conference (GDC) 2016 (2016)
50. Sandy, M., Andersson, J., Barré-Brisebois, C.: Directx: Evolving microsoft's graphics platform. Game Developers Conference 2018 (2018)

51. Scherzer, D., Yang, L., Mattausch, O., Nehab, D., Sander, P.V., Wimmer, M., Eisemann, E.: Temporal coherence methods in real-time rendering. Comput. Graph. Forum **31**(8), 2378–2408 (dec 2012)
52. Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C.R.A., Burgess, J., Liu, S., Dachsbacher, C., Lefohn, A., Salvi, M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In: Proceedings of High Performance Graphics. HPG '17, Association for Computing Machinery, New York, NY, USA (2017)
53. Schollmeyer, A., Schneegans, S., Beck, S., Steed, A., Froehlich, B.: Efficient hybrid image warping for high frame-rate stereoscopic rendering. IEEE Transactions on Visualization and Computer Graphics **23**(4), 1332–1341 (2017)
54. Sitzmann, V., Martel, J.N., Bergman, A.W., Lindell, D.B., Wetzstein, G.: Implicit neural representations with periodic activation functions. In: Proc. NeurIPS (2020)
55. Sun, D., Yang, X., Liu, M.Y., Kautz, J.: Models matter, so does training: An empirical study of cnns for optical flow estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(6), 1408–1423 (2020). https://doi.org/10.1109/TPAMI.2019.2894353
56. Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., Ng, R.: Fourier features let networks learn high frequency functions in low dimensional domains. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 7537–7547. Curran Associates, Inc. (2020)
57. Wang, Z., Chen, J., Hoi, S.C.H.: Deep learning for image super-resolution: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(10), 3365–3387 (2021)
58. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4), 600–612 (2004)
59. Xiang, X., Tian, Y., Zhang, Y., Fu, Y., Allebach, J.P., Xu, C.: Zooming slow-mo: Fast and accurate one-stage space-time video super-resolution. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3370–3379 (June 2020)
60. Xiao, K., Liktor, G., Vaidyanathan, K.: Coarse pixel shading with temporal supersampling. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D '18, Association for Computing Machinery, New York, NY, USA (2018)
61. Xiao, L., Nouri, S., Chapman, M., Fix, A., Lanman, D., Kaplanyan, A.: Neural supersampling for real-time rendering. ACM Trans. Graph. **39**(4) (Jul 2020)
62. Xu, G., Xu, J., Li, Z., Wang, L., Sun, X., Cheng, M.M.: Temporal modulation network for controllable space-time video super-resolution. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6384–6393 (2021)
63. Xu, X., Siyao, L., Sun, W., Yin, Q., Yang, M.H.: Quadratic video interpolation. In: NeurIPS (2019)
64. Yan, L.Q., Mehta, S.U., Ramamoorthi, R., Durand, F.: Fast 4d sheared filtering for interactive rendering of distribution effects. ACM Trans. Graph. **35**(1) (dec 2016)
65. Yang, L., Liu, S., Salvi, M.: A survey of temporal antialiasing techniques. Computer Graphics Forum **39**(2), 607–621 (2020)
66. Yang, L., Nehab, D., Sander, P.V., Sitthi-amorn, P., Lawrence, J., Hoppe, H.: Amortized supersampling. ACM Trans. Graph. **28**(5), 1–12 (Dec 2009)
67. Yang, L., Tse, Y.C., Sander, P.V., Lawrence, J., Nehab, D., Hoppe, H., Wilkins, C.L.: Image-based bidirectional scene reprojection. In: Proceedings of the 2011 SIGGRAPH Asia Conference. pp. 1–10 (2011)
68. Yi, Z., Tang, Q., Azizi, S., Jang, D., Xu, Z.: Contextual residual aggregation for ultra high-resolution image inpainting (2020)
69. Zeng, Z., Liu, S., Yang, J., Wang, L., Yan, L.Q.: Temporally reliable motion vectors for real-time ray tracing. Computer Graphics Forum **40**(2), 79–90 (2021)

70. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 586–595. Computer Vision Foundation / IEEE Computer Society (2018). https://doi.org/10.1109/CVPR.2018.00068