Mob-FGSR: Frame Generation and Super Resolution for Mobile Real-Time Rendering
– – Supplementary Material – –

1 ACCURATE MOTION ESTIMATION

During the frame generation process, we propose a method for estimating the motion of each pixel in adjacent frames using the following equations:

$$p_\alpha = p_1 - \boldsymbol{m}_1 + \frac{\alpha}{2}(\boldsymbol{m}_1 + \boldsymbol{m}_0) + \frac{\alpha^2}{2}(\boldsymbol{m}_1 - \boldsymbol{m}_0), \tag{1}$$

$$p_{1+\alpha} = p_1 + \frac{\alpha}{2}(3\boldsymbol{m}_1 - \boldsymbol{m}_0) + \frac{\alpha^2}{2}(\boldsymbol{m}_1 - \boldsymbol{m}_0), \tag{2}$$

where α represents the time of frame generation, and p_α and $p_{1+\alpha}$ denote the predicted pixel positions for the interpolated and extrapolated frames, respectively. The motion vectors $\boldsymbol{m}_0$ and $\boldsymbol{m}_1$ of the I-frames are used in these calculations. Based on the assumption of quadratic motion (uniform acceleration) between adjacent frames, we derive the detailed formulations as follows.

Referring to the green ball in Fig. 1, we first define the time interval between rendering frames as t. The ball's uniform acceleration is given by:

$$\ddot{\boldsymbol{x}} = \frac{(\boldsymbol{m}_1/t) - (\boldsymbol{m}_0/t)}{t}, \tag{3}$$

$$= \frac{\boldsymbol{m}_1 - \boldsymbol{m}_0}{t^2}, \tag{4}$$

and the initial velocity at I-frame 0 is expressed as:

$$\dot{\boldsymbol{x}}_0 = \frac{\boldsymbol{m}_1 + \boldsymbol{m}_0}{2t}. \tag{5}$$

The position of the green ball, starting from point p_0, can then be calculated by:

$$p_\alpha = p_0 + \dot{\boldsymbol{x}}_0 \cdot \alpha t + \frac{1}{2}\ddot{\boldsymbol{x}} \cdot (\alpha t)^2, \tag{6}$$

$$= p_0 + \frac{\boldsymbol{m}_1 + \boldsymbol{m}_0}{2} \cdot \alpha + \frac{\boldsymbol{m}_1 - \boldsymbol{m}_0}{2} \cdot \alpha^2. \tag{7}$$

However, as the current frame is I-frame 1, we modify the motion to start from point p_1. Representing p_0 as $p_1 - \boldsymbol{m}_1$ yields:

$$p_\alpha = p_1 - \boldsymbol{m}_1 + \frac{\alpha}{2}(\boldsymbol{m}_1 + \boldsymbol{m}_0) + \frac{\alpha^2}{2}(\boldsymbol{m}_1 - \boldsymbol{m}_0), \tag{8}$$

leading to Eq. 1 for interpolation.

To compute the projected point $p_{1+\alpha}$ for the extrapolated frame, we first determine the velocity of the green ball at p_1, summing the change in velocity to its velocity at p_0:

$$\dot{\boldsymbol{x}}_1 = \dot{\boldsymbol{x}}_0 + \ddot{\boldsymbol{x}} \cdot t. \tag{9}$$

The ball's motion is then described as:

$$p_{1+\alpha} = p_1 + \dot{\boldsymbol{x}}_1 \cdot \alpha t + \frac{1}{2}\ddot{\boldsymbol{x}} \cdot (\alpha t)^2, \tag{10}$$

$$= p_1 + (\dot{\boldsymbol{x}}_0 + \ddot{\boldsymbol{x}} \cdot t) \cdot \alpha t + \frac{1}{2}\ddot{\boldsymbol{x}} \cdot (\alpha t)^2. \tag{11}$$

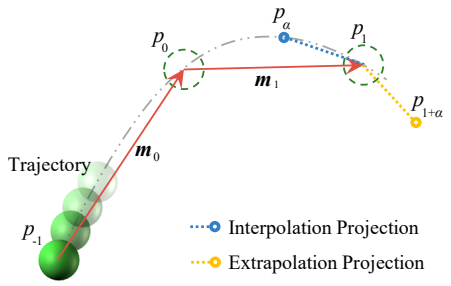


Figure 1: Illustration of the motion estimation process.

Upon inserting variables $\ddot{\boldsymbol{x}}$ and $\boldsymbol{m}_0$, we obtain Eq. 2:

$$p_{1+\alpha} = p_1 + (\frac{\boldsymbol{m}_1 + \boldsymbol{m}_0}{2t} + \frac{\boldsymbol{m}_1 - \boldsymbol{m}_0}{t^2} \cdot t) \cdot \alpha t + \frac{1}{2}\frac{\boldsymbol{m}_1 - \boldsymbol{m}_0}{t^2} \cdot (\alpha t)^2, \tag{12}$$

$$= p_1 + \frac{\alpha}{2}(3\boldsymbol{m}_1 - \boldsymbol{m}_0) + \frac{\alpha^2}{2}(\boldsymbol{m}_1 - \boldsymbol{m}_0). \tag{13}$$

2 DATASET AND TRAINING OF EXISTING DEEP LEARNING METHODS

To evaluate and compare various supersampling methods, we collect a comprehensive dataset from four Unity scenes and two Unreal Engine (UE) scenes. As depicted in Fig. 2, the gaming scenes utilized include: *Street View* (SV), *Meadows* (ME), *Hilly Area* (HA), and *Dragon Park* (DP) from Unity; and *Bunker* (BK) and *Western Town* (WT) from UE. The Unity scenes are rendered using a forward rendering pipeline. Scenes SV and ME feature third-person perspectives, complex character motions, and rich texture details. Scenes HA and DP provide a free camera view, presenting broad scene motion and rich environmental elements. The UE scenes, rendered using a deferred pipeline, are captured from a third-person viewpoint.

For each environment, we collect 8 video clips, with each video containing 500 frames. The first 350 frames of each video are allocated for training deep learning-based models, like NSR [Xiao et al. 2020] and ExtraNet [Guo et al. 2021], while the remaining 150 frames serve as a test set. Super-resolution methods take input at 540P and output at 1080P. The frame generation process is always running at 1080P. To create alias-free ground-truth images as references, we employ 9× MSAA when collecting the dataset.

We used official codes that are publicly available or provided by the authors to implement the methods of MNSS [Yang et al. 2023] and ExtraNet [Guo et al. 2021]. Additionally, we replicate the NSR [Xiao et al. 2020] method based on an open-source implementation[1]. These methods are trained using the input specifications

[1]https://github.com/IMAC-projects/NSRR-PyTorch

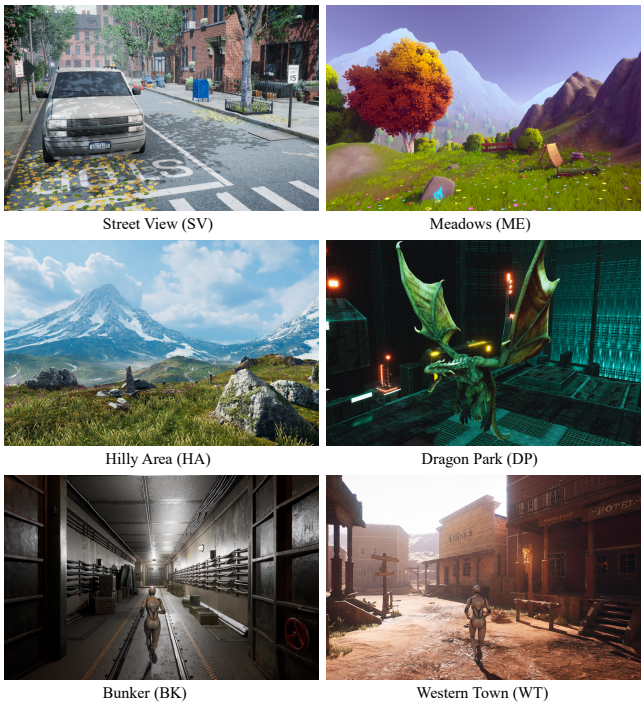


Figure 2: Illustration of representative scenes of our dataset.

and training strategies outlined by the authors in their respective publications, resulting in trained models that are ready for testing.

3 UNIFORM VELOCITY / UNIFORMLY ACCELERATED MOTION ASSUMPTIONS

Our method for estimating motion between adjacent frames is based on the assumption of uniformly accelerated motion. This more general assumption accurately captures the motion of pixels for both interpolation and extrapolation. Another viable solution for motion reconstruction is the uniform velocity assumption, which implies constant speed linear motion. The uniform velocity motion can be expressed as follows:

$$p_\alpha = p_1 - (1-\alpha)\cdot \boldsymbol{m}_1, \tag{14}$$

$$p_{1+\alpha} = p_1 + \alpha \cdot \boldsymbol{m}_1. \tag{15}$$

These formulas are similar to Eqs. 1 and 2, which predict the position of pixels in interpolated or extrapolated frames.

Using SV and DP scene data, we compare the results of methods based on the assumptions of uniform velocity and uniformly accelerated motion. Tab. 1 presents a quantitative comparison of frame generation for both approaches. The models that use the uniform velocity motion assumption (Ours-I' and Ours-E') perform slightly worse than our final models (Ours-I and Ours-E). The error maps between the outputs of these two methods and their ground truth images are shown in Fig. 3, indicating that our final model has smaller errors. Both methods have nearly identical runtimes, with 2.21ms and 2.20ms for Ours-I and Ours-I', respectively, and 1.92ms for both Ours-E and Ours-E'. Given these results, we recommend adopting the uniformly accelerated motion assumption method.

Table 1: Quantitative comparison between methods using uniform velocity assumption (Ours-I' and Ours-E') and those employing uniformly accelerated motion assumption (Ours-I and Ours-E), on the SV and DP scenes.

	Method	PSNR↑		SSIM↑	
		SV	DP	SV	DP
Interpolation	Ours-I'	27.26	32.45	0.863	0.912
	Ours-I	**27.57**	**32.77**	**0.878**	**0.922**
Extrapolation	Ours-E'	25.88	30.03	0.842	0.898
	Ours-E	**26.60**	**30.65**	**0.854**	**0.908**

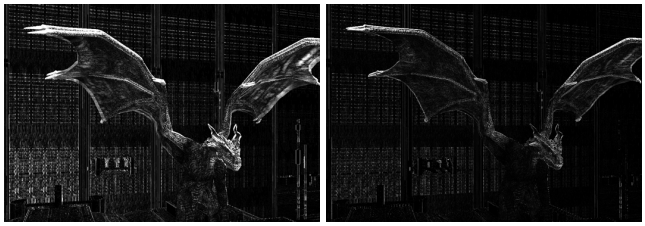


Figure 3: Visual comparison of error maps generated by contrasting the results from methods assuming uniform velocity and those assuming uniformly accelerated motion, with the ground truth.

4 DISOCCLUSION FILLING

Addressing the disocclusion issue is crucial in frame extrapolation to minimize ghosting artifacts. We evaluate various disocclusion filling methods in the test environment established in Sec. 4 of the main paper. These methods include background duplication (BD) [Andreev 2010], depth-based filtering (DBF) [Schollmeyer et al. 2017], and our method that utilizes MVs from the previous I-frame for disocclusion filling. In our tests, we observe that the BD and DBF methods generally yield over-smoothed results (see Fig. 4 (a)). These methods involve multiple high-cost blurring passes, resulting in significant latency (more than 3ms on the test device). In contrast, our approach rapidly repairs MVs and uses subsequent image warping for disocclusion filling, which achieves minimal latency (about 0.2ms) and is ideally suited for high frame rate applications.

5 LEARNING LUT AND THRESHOLDS

In Sec. 3.5 and Sec. 4 of the main paper, we introduce learned resampling weights based on a Lookup Table (LUT) for repeated image warping on mobile platforms. This data-driven strategy is designed to learn the most effective filters for repeated sampling. Using the evaluation environment established in Sec. 4 of the main paper, we train a multilayer perceptron (MLP) to determine pixel weights based on the positions of sampling points. These learned weights are subsequently used to perform repeated image warping, and we optimize the MLP to predict sampling weights that achieve the highest quality of warped images (see Fig. 4 (b) for a representative comparison). After training, the weights derived from the MLP are stored in a 32×32×16 LUT for fast access. The MLP does not

Table 2: Runtime performance on the Qualcomm Snapdragon 8 Gen 2 SoC.

	Frame generation (720P)		Frame generation & SR (1080P)		
Modules	Ours-I	Ours-E	Ours-SR	Ours-ISR	Ours-ESR
MVs Splatting	0.94	0.79	-	0.52	0.41
Refinement	0.73	0.65	-	0.38	0.35
Disocc. Filling	-	0.29	-	-	0.23
Warping	0.96	0.74	1.54	1.62	1.27
Blending	0.15	-	0.58	0.28	-
Total	2.78ms	2.47ms	2.12	2.80ms	2.26ms

require a specialized design; we employ a straightforward architecture consisting of four layers with neuron numbers [16, 32, 64, 16] and use the tanh activation function. We utilize a simple (1-SSIM) as the loss function. The training process is conducted using the PyTorch deep learning framework, utilizing the Adam optimizer with a learning rate of 10^{-3}, betas of (0.9, 0.999), and eps set to 10^{-4}. Training typically converges within about 20 minutes on an NVIDIA GeForce RTX 3090 GPU. Some of the trained LUTs will be available on the project page.

For learning the optimal parameters T_D and T_B for validating history pixels and reusing previous SR images, we employ the same training strategy. The thresholds T_D and T_B are set as learnable parameters to be optimized within the deep learning framework. All other training settings, including the learning rate and loss function, are maintained consistently. Training these parameters also takes about 20 minutes on the same platform.

6 PERFORMANCE ON ADDITIONAL HARDWARE

Due to length constraints, our main paper only covers the runtime performance of our model on the Qualcomm Snapdragon 8 Gen 3 processor. This section expands the evaluation to include additional mobile processor, the Qualcomm Snapdragon 8 Gen 2 (released in 2022). These processors, commonly used in smartphones, tablets, and other mobile devices, are chosen as suitable testing platforms. Tab. 2 presents an assessment of runtime performance for the Snapdragon 8 Gen 2. Our method consistently delivers fast processing speeds and robust performance on older hardware. We will conduct future tests on other mobile processors and update the project homepage[2] with new results.

7 ANDROID DEMO

To demonstrate the practical application of our proposed method, we have developed an Android real-time rendering application. This application allows for the verification of model runtimes and assessment of reconstructed image quality. Users can test the application by downloading it from our project homepage and installing it on their Android smartphones. We recommend testing the app on Android devices equipped with the two Qualcomm processors mentioned above, as it has only been tested on these platforms so far. Furthermore, due to energy optimization policies implemented by smartphone manufacturers, CPU and GPU frequency may be limited for unrecognized applications. Users may need to add our application to the whitelist in their smartphones' developer mode to ensure optimal performance.

Fig. 5 presents the key features of our Android application. In an urban street view, we use a looped camera motion path and implement resolution-related no-operation computational tasks to emulate intensive rendering environments. Sub-figure (a) shows the real-time rendering performance without any supersampling enabled, achieving a frame rate of about 22 FPS. Our interpolation model improved frame rate from 22 to 48 FPS by interpolating 2 frames between adjacent pairs, as shown in sub-figure (b). Similarly, sub-figure (c) shows how our extrapolation model improved the frame rate to 54 FPS. Both interpolated (b) and extrapolated (c) frames maintain the same quality as the original rendering (a). Sub-figures (d) and (e) show the combined effects of frame interpolation and super-resolution, as well as frame extrapolation and super-resolution, resulting in significant improvements at frame rates of 113 FPS and 117 FPS, respectively.

[2]https://mob-fgsr.github.io/

REFERENCES

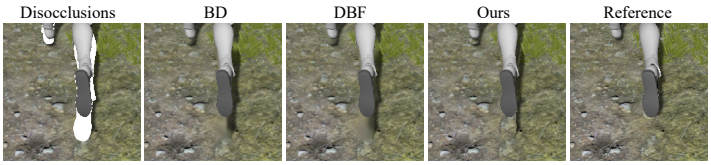
Dmitry Andreev. 2010. Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for" free". In *ACM SIGGRAPH 2010 Talks*. 1–1.

Jie Guo, Xihao Fu, Liqiang Lin, Hengjun Ma, Yanwen Guo, Shiqiu Liu, and Ling-Qi Yan. 2021. Extranet: Real-time extrapolated rendering for low-latency temporal supersampling. *ACM Transactions on Graphics (TOG)* 40, 6 (2021), 1–16.

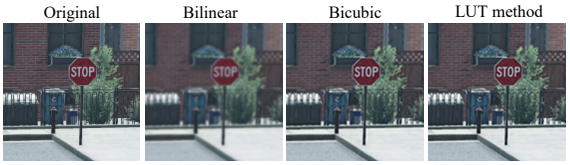
Andre Schollmeyer, Simon Schneegans, Stephan Beck, Anthony Steed, and Bernd Froehlich. 2017. Efficient hybrid image warping for high frame-rate stereoscopic rendering. *IEEE Transactions on Visualization and Computer Graphics* 23, 4 (2017), 1332–1341.

Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural supersampling for real-time rendering. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 142–1–12.

Sipeng Yang, Yunlu Zhao, Yuzhe Luo, He Wang, Hongyu Sun, Chen Li, Binghuang Cai, and Xiaogang Jin. 2023. MNSS: Neural Supersampling Framework for Real-Time Rendering on Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics* (2023), 1–14.



(a) Disocclusion filling



(b) Repeated resampling

Figure 4: Representative examples of disocclusion filling and repeated resampling results using various methods. (a) We demonstrate disocclusion filling with background duplication (BD), depth-based filtering (DBF), and our method. (b) We show examples of sixfold resampling using bilinear, bicubic, and learned LUT methods, with half-pixel shifts in both the horizontal and vertical axes.

(a) Without Supersampling

(b) Our Interpolation Model

(c) Our Extrapolation Model

(d) Our Interpolation & SR Model

(e) Our Extrapolation & SR Model

Figure 5: Demonstration of the Android app in a street view: (a) rendering without supersampling; (b) using our interpolation model; (c) using our extrapolation model; (d) interpolation combined with super resolution model; (e) extrapolation combined with super resolution model. The top-right corner of each image displays the rendering resolution and frame rate.