Kernel-Based Frame Interpolation for Spatio-Temporally Adaptive Rendering (Supplementary Document)

Karlis Martins Briedis
DisneyResearch|Studios
Zürich, Switzerland
ETH Zürich
Zürich, Switzerland
karlis.briedis@inf.ethz.ch

Abdelaziz Djelouah
DisneyResearch|Studios
Zürich, Switzerland
aziz.djelouah@disneyresearch.com

Raphaël Ortiz
DisneyResearch|Studios
Zürich, Switzerland
raphael.ortiz@disneyresearch.com

Mark Meyer
Pixar Animation Studios
Emeryville, California, USA
mmeyer@pixar.com

Markus Gross
DisneyResearch|Studios
Zürich, Switzerland
ETH Zürich
Zürich, Switzerland
grossm@inf.ethz.ch

Christopher Schroers
DisneyResearch|Studios
Zürich, Switzerland
christopher.schroers@disneyresearch.com

ACM Reference Format:
Karlis Martins Briedis, Abdelaziz Djelouah, Raphaël Ortiz, Mark Meyer, Markus Gross, and Christopher Schroers. 2023. Kernel-Based Frame Interpolation for Spatio-Temporally Adaptive Rendering (Supplementary Document). In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings), August 6–10, 2023, Los Angeles, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3588432.3591497

1 KERNEL-BASED FRAME SYNTHESIS

1.1 Numerically Stable Softmax Splatting

When performing motion compensation with forward warping, also known as splatting, each pixel in the source image is added to the target, by using weighted averaging to handle mapping ambiguities. Formally, the value of the splatted image $\mathbf{I}$, given an optical flow map $\mathbf{f}$, for an output location $\boldsymbol{y}$ on the image plane Ω for softmax splatting [Niklaus and Liu 2020] can be defined as

$$\mathcal{S}(\mathbf{I})[\boldsymbol{y}] = \left(\sum_{\boldsymbol{x}\in\Omega} w(\boldsymbol{x},\boldsymbol{y}) \cdot \mathbf{I}[\boldsymbol{x}]\right) \cdot \left(\sum_{\boldsymbol{x}\in\Omega} w(\boldsymbol{x},\boldsymbol{y}) + \varepsilon\right)^{-1} \quad (1a)$$

$$w(\boldsymbol{x},\boldsymbol{y}) = \exp(\mathbf{Z}[\boldsymbol{x}]) \cdot k(\boldsymbol{x} + \mathbf{f}[\boldsymbol{x}] - \boldsymbol{y}) \quad (1b)$$

with a given per-pixel weighting/importance map $\mathbf{Z}$ and kernel k, centered around the displacement location $\boldsymbol{x} + \mathbf{f}[\boldsymbol{x}]$. As in [Briedis et al. 2021], we use a bilinear kernel.

If implemented directly as described, it has several numerical issues when the sum of weights $\sum_{\boldsymbol{x}\in\Omega} w(\boldsymbol{x},\boldsymbol{y})$ is very large or small. For large weights the output is affected by the floating point arithmetic round off error, for small weights, e.g. when the output location is far from any displacement center and all $k(*) << 1$, in addition to the round off error, the normalization has a non-negligible value shift from the ε factor that is used to prevent division by zero. Such color shifts are especially troublesome for our kernel-based synthesis method, as the darkening breaks the linearity assumption of the splatting step.

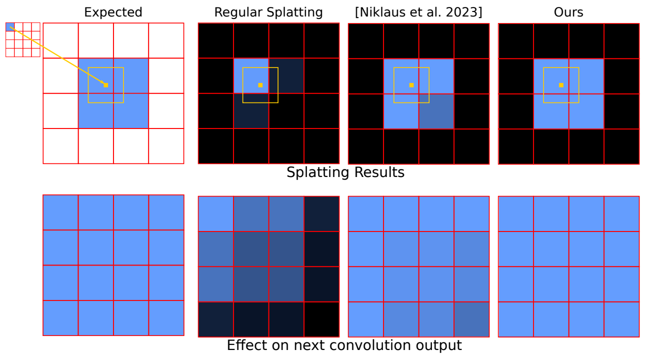

Figure 1: A toy example showing normalized softmax splatting of a single pixel. A bilinear kernel is used thus the target spans across 4 pixels (in yellow, offset has been increased for the visualization). Second row is showing the reconstructed image after applying a 3x3 uniform kernel while ignoring boundary pixels with no contributions. It can be observed that the prior methods can introduce color shift.

To resolve both issues, concurrently with [Niklaus et al. 2023] and as commonly done in deep learning frameworks for the regular softmax normalization [Goodfellow et al. 2016], we use the translational invariance property of softmax and subtract the maximum weight. To further improve stability due to small splatting kernel weights, we take logarithm of the kernel and move it inside the exponentiation. With that, we rewrite Eq. 1 as

Table 1: Model parameters

Module	NFIRC [Briedis et al. 2021]	Ours
Flow network	$3.895M$	$3.895M$
Full feature encoder	$206K$	$73K$
Partial feature encoder	$20K$	$13K$
Weight map estimator	$118K$	$118K$
GridNet	$3.272M$	$2.899M$
Key/Query estimation	-	$2.528K$
Total	$7.543M$	$7.002M$

$$\mathcal{S}(\mathbf{I})[\boldsymbol{y}] = \left(\sum_{\boldsymbol{x}\in\Omega} w^*(\boldsymbol{x},\boldsymbol{y}) \cdot \mathbf{I}[\boldsymbol{x}]\right) \cdot \left(\sum_{\boldsymbol{x}\in\Omega} w^*(\boldsymbol{x},\boldsymbol{y}) + \varepsilon\right)^{-1} \tag{2a}$$

$$w^*(\boldsymbol{x},\boldsymbol{y}) = \exp(\mathbf{Z}[\boldsymbol{x}] + \log(k(\boldsymbol{x} + \mathbf{f}[\boldsymbol{x}] - \boldsymbol{y})) - \mathbf{m}(\boldsymbol{y})) \tag{2b}$$

$$\mathbf{m}(\boldsymbol{y}) = \max_{\boldsymbol{x}\in\Omega}(\mathbf{Z}[\boldsymbol{x}] + \log(k(\boldsymbol{x} + \mathbf{f}[\boldsymbol{x}] - \boldsymbol{y}))). \tag{2c}$$

By subtracting the maximum term, it ensures that

$$\max_{\boldsymbol{x}\in\Omega} w^*(\boldsymbol{x},\boldsymbol{y}) = exp(0) = 1 \tag{3a}$$

$$1 \leq \sum_{\boldsymbol{x}\in\Omega} w^*(\boldsymbol{x},\boldsymbol{y}) \leq |\Omega| \tag{3b}$$

for every pixel $\boldsymbol{y}$ with at least a single non-zero weight contribution, making the division stable and unaffected by the ε. In practice, it can be implemented in two passes - in the first pass $\mathbf{m}(\boldsymbol{y})$ is estimated for each target pixel $\boldsymbol{y}$ by performing maximum forward warping, and in the second pass performing shifted softmax splatting. Unlike the method of Niklaus *et al.* [2023], our method is stable even for small splatting kernel coefficients as demonstrated in Figure 1.

1.2 Model Architecture

Baseline. For our baseline model, we follow [Briedis et al. 2021] and refer the reader to the respective article for more details. We replace the shifted *ELU* [Clevert et al. 2016] weighting with softmax splatting [Niklaus and Liu 2020], and use our kernel-based frame synthesis approach. To compensate for the increased computations at full level scale, we reduce the output dimensions of full and partial context encoders to $16, 32, 64$ and $8, 16, 32$ channels with no noticeable decrease in quality.

Query and Scaling Estimation. For the initial estimate, we use a GridNet [Fourure et al. 2017] as in [Niklaus and Liu 2020] with $[32, 64, 96]$ channels on each input, and set the output number of channels to 16. The queries and scalings are predicted from this initial estimate with networks that consist of $\boldsymbol{rc}12\boldsymbol{rc}12$ and two $\boldsymbol{rc}8\boldsymbol{rc}1$, where R is a ReLU activation and $\boldsymbol{ck}$ is 1×1 convolution with k output channels. Keys and biases are estimated with equal networks to queries and keys, but with separate parameters.

Model Size. In Table 1 we show the number of parameters of the different components compared to [Briedis et al. 2021]. The compact model size allows to interpolate $4K$ content even on low-memory hardware and it takes $2.53{\pm}0.01s$ and $< 23GB$ to interpolate a single 3840×2160 frame on a *NVIDIA RTX A6000* GPU.

1.3 Extended linear to sRGB color transform

We use the following formula to perform the linear to extended sRGB color transform for each of the color channels x.

$$\begin{cases} lin2sRGB(x) & \text{if } x \leq 1 \\ 0.38278 \cdot \log(x - 0.12922) + 1.05296 & \text{otherwise} \end{cases}$$

1.4 Ablation Study Implementation Details

In this subsection we provide implementation details of our ablation study variants.

Dynamic Kernel Prediction Methods. We simply set $a^i_{\boldsymbol{y}} = 1$, $b^i_{\boldsymbol{x}} = 0$, or rewrite the weight computation equation as

$$w^i_{\boldsymbol{yx}} = (a^i_{\boldsymbol{y}})^2(\mathbf{q}_{\boldsymbol{y}})^T\mathbf{k}^i_{\boldsymbol{x}} + b^i_{\boldsymbol{x}}, \tag{4}$$

Direct Prediction. We use [Briedis et al. 2021] trained with our extended sRGB color transform to support HDR images.

Affinity-based Kernel Prediction. To adapt the kernel estimation using affinity of neural features [Işık et al. 2021] to frame interpolation, we concatenate all top-level splatted features, same as the inputs to the GridNet [Fourure et al. 2017] in our approach, and use them as the input for a UNet [Ronneberger et al. 2015] with the same size as in the denoising approach. Instead of predicting a single set of features/bandwidth parameters $\mathbf{f}^k_{xyt}$, a^k_{xyt}, c^k_{xyt}, we estimate two sets for each of the keyframes and use them to compute per-splat w^k_{xyuvt}. These weights are then applied for each frame, summed, normalized with the sum of the weights, and applied with increased dilations as in the denoising approach.

Direct Kernel Prediction. We directly predict two sets of 11×11 per-pixel kernels from the output of the GridNet with two $\boldsymbol{rc}64\boldsymbol{rc}121$ convolutional layers.

Direct Multi-Scale Kernel Prediction. Following the single scale approach, we apply direct kernel prediction on 3 levels of scale, using $16, 32, 48$ feature outputs from GridNet as the input for kernel estimator. Additionally, we predict weight parameter α_l for each scale l with a $\boldsymbol{rc}12\boldsymbol{rc}1$ layer, upsample all of the inputs and use softmax weighting over $\{\alpha_l\}$ to merge all bilinearly upsampled outputs.

1.5 Additional Results

In Figure 2 we provide a visual comparison of the method from our ablation study, and in Figure 3 show the error distributions of these methods as *kernel density estimate* plots, showing that our method has higher density towards better scores.

In Figure 4 we show equivalent plots for the comparisons with the best-performing prior methods.

1.6 Multi-channel interpolation with prior methods

To interpolate channels for which NFIRC [Briedis et al. 2021] was not trained, we estimate motion and weighting coefficients with the original image and run the pyramid extraction, warping, and frame synthesis on the additional channels. For the alpha channel

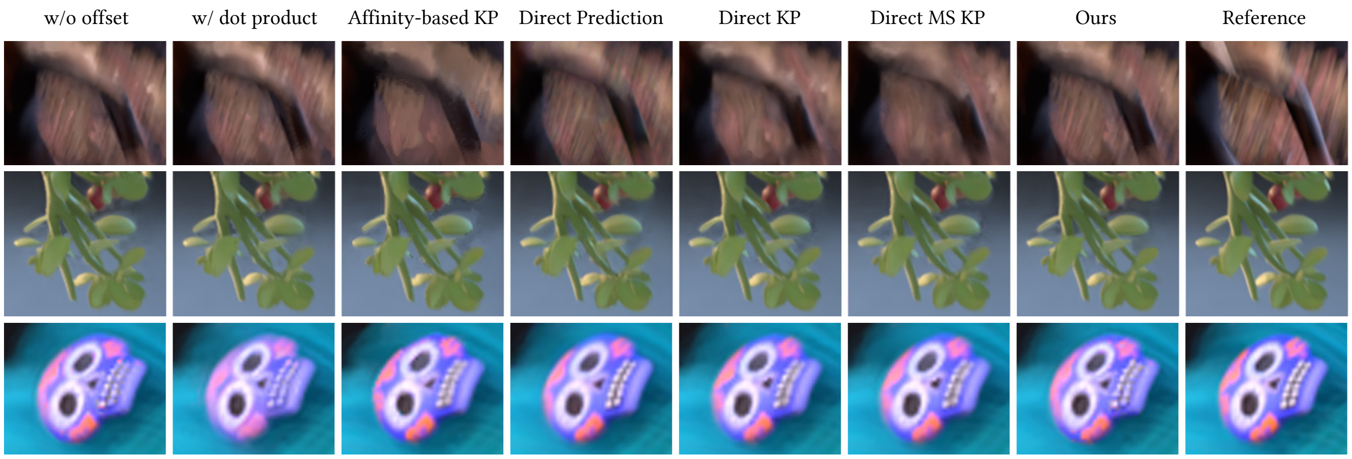

Figure 2: Visual comparison of the methods from our ablation study. © 2023 Disney, © 2023 Disney / Pixar

3× repeated values are interpolated with the mean value across channel using as the final output.

2 ADAPTIVE INTERPOLATION

2.1 Blender Runtime Experiments

To evaluate the latency added by an additional auxiliary buffer rendering pass, we extend BLENDER's CYCLES physically-based renderer to record per-tile rendering time and an option to render only feature buffers. In Table 3, we detail CPU and GPU times for different proportions of rendered pixels. We see that the adaptive strategy introduces a negligible overhead while achieving significant performance improvements.

Rendering Details. We choose 3 shots from a recent movie CHARGE that are publicly accessible and have medium motion - 010_0050, 040_0040, and 060_0130. We base our CYCLES adaptations on `v3.5.0` pre-release commit `1a986f7e`.

As the shots are not originally made for the physically-based renderer CYCLES, we apply shot modifications as described in Table 2. When rendering the buffers pass, ray tracing is terminated at the intersection where the renderer records denoising passes.

To compute the constant ramp up costs for loading a shot that is present even if rendering only a single tile, we approximate it by rendering a 1*spp* variant of the each shot.

Adaptive Runtime Computation. To approximate the runtime for adaptive interpolation, we rescale the runtime of each tile to sum up to $total\_runtime - ramp\_up\_runtime$, and sum up the time it takes to render each of the requested tiles with $ramp\_up\_runtime$. For the GPU time computation, all needed δ_t^k are stored in memory during the processing. They are in 1/16 resolution thus for a 96-frame sequence only 40*MB* are used (1704 maps due to boundaries with 51×120px at 32 bits). Input/Output costs are excluded from the measurements and all images are expected to be stored in the RAM.

2.2 Fixed Interval Interpolation

To obtain the fixed interval interpolation results, we compute the minimal number of keyframes needed to obtain the chosen ratio of rendered pixels, and place them at evenly spaced temporal positions in $[t_{start}, t_{end}]$, rounding to the nearest integer. We then use recursive interpolation, *i.e.* between every two consecutive keyframes, we interpolate the middle frame, set it as a new keyframe, and repeat the process until all frames are set as keyframes.

Table 2: Applied Blender Shot Modifications

`scene.render`	
`resolution_x`	1920
`resolution_y`	804
`use_motion_blur`	`False`
`engine`	`CYCLES`
`scene.cycles`	
`tile_size`	64
`device`	`CPU`
`samples`	1024
`use_adaptive_sampling`	`True`
`adaptive_min_samples`	32
`adaptive_threshold`	0.01
`use_denoising`	`True`
`denoiser`	`OPENIMAGEDENOISE`

2.3 Implementation Details

In the Listing 1 we show the architecture of the implicit error prediction model. On the top 3 levels (the first layer and after pooling layers) we concatenate the inputs with the warped partial context features and their binary masks. In the Listing 2 we show the used residual block. In the top level we input intermediate frame motion magnitude, divided by 256 and clampped to [0, 2].

Listing 1: Architecture of interval prediction model

```
Sequential(
    Conv2d(27, 32, kernel_size=3)
    ResBlock(32, 64)
    MaxPool2d(kernel_size=2, stride=2)
```

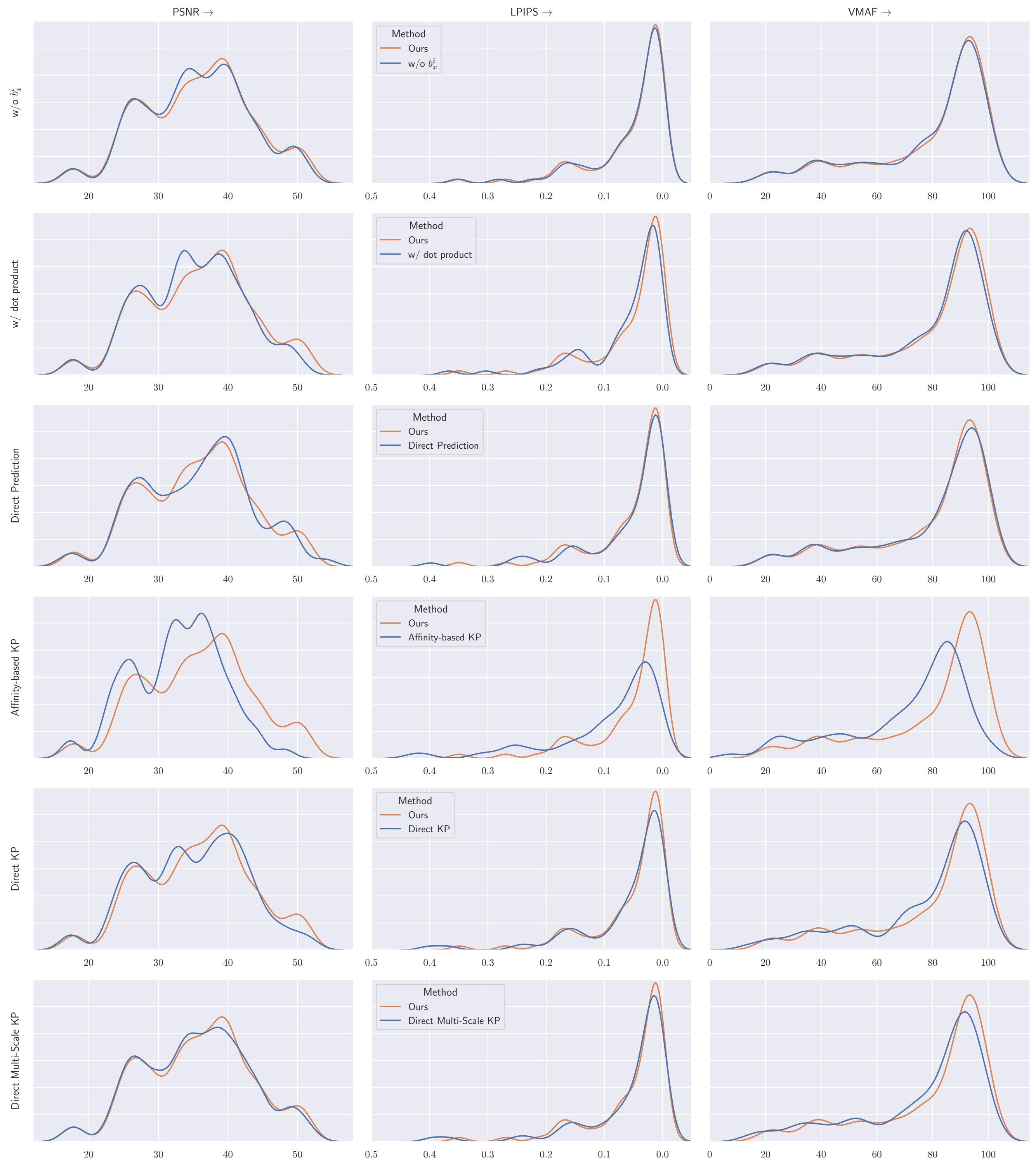

Figure 3: Distribution of the error for the different methods of our ablation study. Each column reports a different metric (PSNR, LPIPS, VMAF). The *y-axis* shows the probability density estimate. The best performing method should have the distribution more to the right towards the better scores.

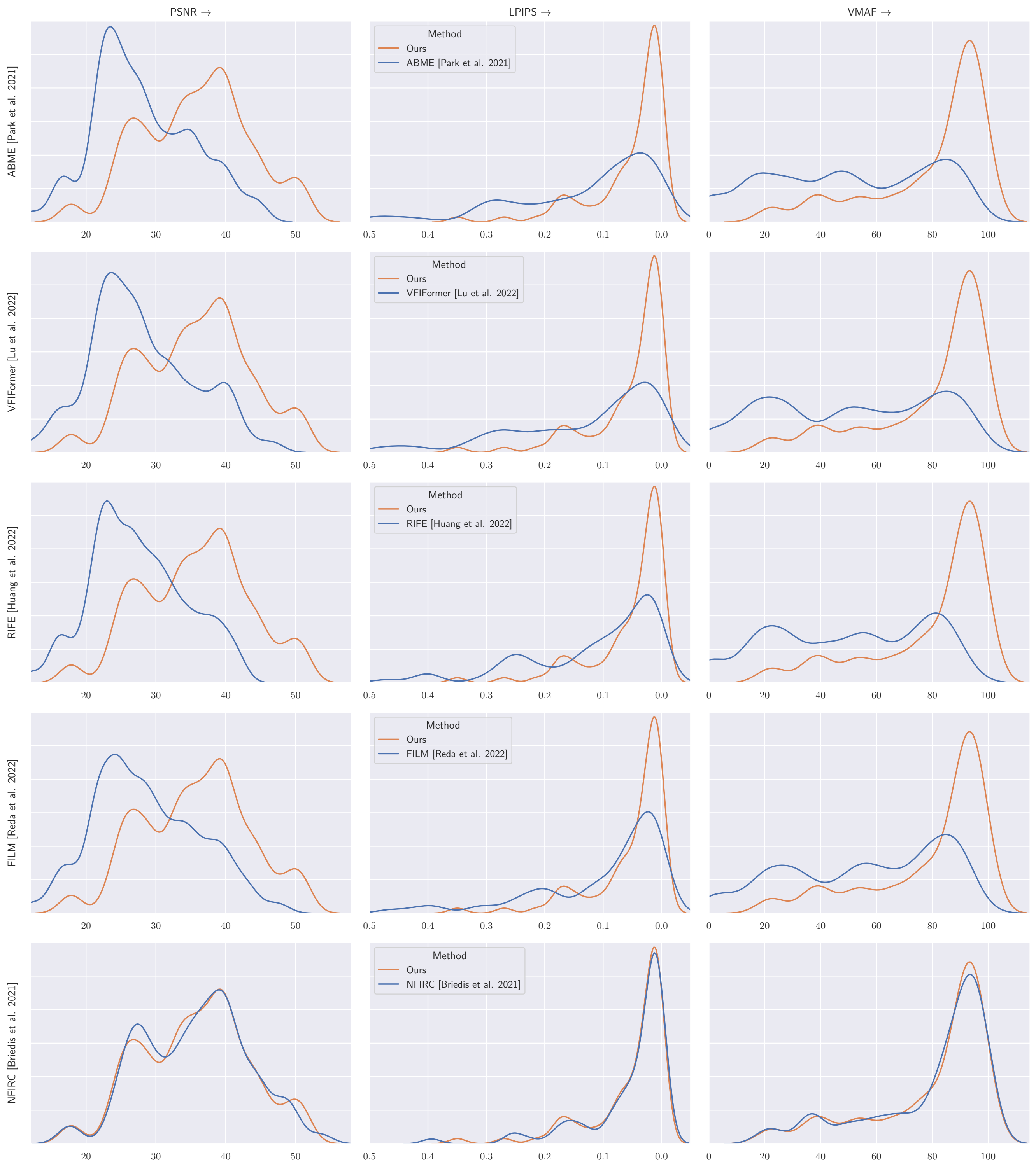

Figure 4: Distribution of the error for the different prior methods compared to our method. Each column reports a different metric (PSNR, LPIPS, VMAF). The *y-axis* shows the probability density estimate. The best performing method should have the distribution more to the right towards the better scores.

Table 3: Runtime breakdown. Computed mean per 1 frame and averaged over 3 shots.

Method	Rendered Ratio	CPU time, *min*			GPU time, *min*			PSNR, dB
		Buffers	Beauty	**Total**	Interval Estimation	Interpolation	**Total**	
Full Render	100%	-	109.5	109.5	-	-	-	-
Adaptive Interval	48.5%	15.7	62.5	78.2	0.09	0.03	0.12	43.78
	32.6%	15.7	40.8	56.5		0.03	0.12	41.95
	24.7%	15.7	30.4	46.1		0.03	0.12	40.94
	19.5%	15.7	24.0	39.7		0.03	0.12	40.31
	16.5%	15.7	20.1	35.8		0.03	0.12	39.81
Fixed Interval	50.8%	7.8	67.2	75.0	-	0.02	0.02	41.03
	34.4%	10.2	45.8	56.1		0.02	0.02	39.79
	26.1%	11.5	34.7	46.2		0.02	0.02	37.89
	21.0%	12.3	27.9	40.2		0.03	0.03	37.57
	18.0%	12.8	24.1	37.0		0.03	0.03	37.13

```
    ResBlock(114, 128)
    MaxPool2d(kernel_size=2, stride=2)

    ResBlock(202, 128)
    MaxPool2d(kernel_size=2, stride=2)

    ResBlock(128, 128)
    MaxPool2d(kernel_size=2, stride=2)

    ResBlock(128, 32)
    Conv2d(27, 32, kernel_size=3)
    Sigmoid()
)
```

Listing 2: Residual block architecture

```
ResBlock(in, out)(
  ReLU(
    Sequential(
      Conv2d(in, out, kernel_size=3)
      ReLU()
      Conv2d(out, out, kernel_size=3)
    ) + (
      if (in == out)
        Identity()
      else
        Conv2d(in, out, kernel_size=1)
    )
  )
)
```

REFERENCES

Karlis Martins Briedis, Abdelaziz Djelouah, Mark Meyer, Ian McGonigal, Markus Gross, and Christopher Schroers. 2021. Neural Frame Interpolation for Rendered Content. *ACM Trans. Graph.* 40, 6, Article 239 (dec 2021), 13 pages. https://doi.org/10.1145/3478513.3480553

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1511.07289

D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Tremeau, and C. Wolf. 2017. Residual conv-deconv grid network for semantic segmentation. *arXiv preprint arXiv:1707.07958* (2017).

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Mustafa Işık, Krishna Mullia, Matthew Fisher, Jonathan Eisenmann, and Michaël Gharbi. 2021. Interactive Monte Carlo Denoising Using Affinity of Neural Features. *ACM Trans. Graph.* 40, 4, Article 37 (jul 2021), 13 pages. https://doi.org/10.1145/3450626.3459793

Simon Niklaus, Ping Hu, and Jiawen Chen. 2023. Splatting-Based Synthesis for Video Frame Interpolation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 713–723.

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

O. Ronneberger, P. Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI) (LNCS, Vol. 9351)*. Springer, 234–241. http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a (available on arXiv:1505.04597 [cs.CV]).